UNIVERSIDAD DE SEVILLA
FACULTAD DE INFORMATICA Y ESTADISTICA

TUTOR: D. FRANCISCO HERNANDEZ RODRIGUEZ

PROYECTO FIN DE CARRERA

HERRAMIENTA DE AYUDA A LA DOCENCIA DE INFORMATICA GRAFICA: EL PROBLEMA DE LA VISIBILIDAD DE GRAFICOS 3D Y SU RESOLUCION MEDIANTE EL ALGORITMO Z-BUFFER

ANTONIO JOSE TORRES RODRIGUEZ DE ALMANSA

SEVILLA, 1998

TUTOR: D. FRANCISCO HERNÁNDEZ RODRÍGUEZ

DEPARTAMENTO DE INGENIERÍA DEL DISEÑO
FACULTAD DE INFORMÁTICA Y ESTADÍSTICA
UNIVERSIDAD DE SEVILLA

PROYECTO FIN DE CARRERA

HERRAMIENTA DE AYUDA A LA DOCENCIA DE INFORMÁTICA GRÁFICA:
EL PROBLEMA DE LA VISIBILIDAD DE GRÁFICOS 3D
Y SU RESOLUCIÓN MEDIANTE EL ALGORITMO Z-BUFFER

ANTONIO JOSÉ TORRES RODRÍGUEZ DE ALMANSA
SEVILLA, SEPTIEMBRE 1998

A *Paqui* y a mis padres,
por el apoyo moral que
me han ofrecido siempre.

INDICE

INDICE

1
INTRODUCCIÓN

1. INTRODUCCIÓN

En este primer capítulo se realiza una breve introducción histórica al problema de la representación gráfica concretando en la visibilidad de los gráficos 3D, tema central de esta obra. Además, se expondrán los objetivos del proyecto y una muestra de la organización del proyecto para que el lector pueda localizar fácilmente la información que crea necesaria.

1.1. INTRODUCCIÓN

Ya desde la Antigüedad, el hombre ha intentado representar todo aquello que puede observar con sus propios ojos. Los prehistóricos, por ejemplo, dibujaban animales en sus cuevas con manchas de pintura creadas a partir del barro y, asimismo, cambiando de tonalidades intentaban que aquello que habían dibujado representara con mayor o menor fiabilidad lo que habían visto.

Sin embargo, no fue hasta la época renacentista que el hombre se dio cuenta que para que una imagen representara la realidad que estaba observando faltaba algo tan importante como es la profundidad. Si se observan los dibujos y pinturas de los egipcios, puede verse que éstos no dibujan los objetos o las personas de una manera realista sino más bien *idealista* (a mayor poder, mayor tamaño), con lo que el concepto de profundidad a la hora de representar algo quedaba fuera de lugar.

Consecuencia de ello, para resolver el problema de la profundidad se desarrolló la Perspectiva, con lo que cualquier escena cotidiana podía representarse de forma similar a como lo hace hoy en día una cámara fotográfica mostrando la realidad tal como es.

Sirva este preámbulo para indicar que, ya en los primeros tiempos del uso de los ordenadores en el campo gráfico, una cuestión que se planteaba era la posibilidad de representar la realidad de una manera fotográfica. No obstante esto entrañaba grandes problemas. Primero, existía la imposibilidad de pintar un gran número de colores en la pantalla, ya que la técnica por aquel entonces se limitaba a pantallas monocromas. Después, se planteó tratar a los objetos como un conjunto de líneas formando mallas de polígonos y al usar un único color surgió la cuestión que se está tratando en esta obra. Esto es, con el uso de imágenes compuestas de superficies poligonales aparecieron multitud de caras que se tapaban unas a otras enmarañando la pantalla y anulando totalmente la comprensión del objeto trazado.

Es, por tanto, cuando los científicos e informáticos se plantean la resolución del problema de eliminación de las líneas y superficies ocultas de objetos. Por supuesto, debido a la capacidad de los ordenadores de aquella época, ciertas técnicas quedaban fuera del alcance práctico, aunque en la actualidad han sido recuperadas dado el espectacular avance de la Informática en estos años.

Entre estas técnicas, métodos y algoritmos se encuentra el tratado en esta obra, el algoritmo Z-Buffer o algoritmo del buffer de profundidades, que quizás sea el que mayor implementación está teniendo en la actualidad, cuando, en un principio, fue arrinconado debido a su alto consumo de memoria.

1.2. OBJETIVOS DEL PROYECTO

En primer lugar, se trata de una *herramienta de ayuda a la docencia de informática gráfica*, es decir, de una aplicación diseñada con el fin específico de servir de apoyo en clases y exposiciones relacionadas con el tema de la Infografía. Como tal, no debe considerarse como un programa de carácter genérico debido a sus limitaciones.

Por otra parte, el proyecto se centra en *el problema de la visibilidad de gráficos 3D y su resolución mediante el algoritmo Z-Buffer*. Esto es, se quiere resolver un problema particular mediante una técnica concreta. El hecho de utilizar un único algoritmo para la resolución del problema es fundamental dado el carácter pedagógico expuesto en el párrafo anterior.

1.3. ORGANIZACIÓN DEL PROYECTO

El proyecto está compuesto por un índice, 11 capítulos y un anexo.

Este capítulo 1, "*Introducción*", trata la historia del problema de la visibilidad de los gráficos 3D así como de los objetivos del proyecto y la organización del mismo. El capítulo 2, "*Eliminación de superficies ocultas*", muestra brevemente algunos de los algoritmos de eliminación de superficies ocultas y una posible clasificación de diversos algoritmos relacionados con el problema.

El capítulo 3, "*Descripción del algoritmo Z-Buffer. Especificaciones del proyecto*", describe cómo es el algoritmo Z-Buffer y muestra las especificaciones recibidas para el desarrollo del proyecto. El capítulo 4, "*Soluciones adoptadas del proyecto*", cubre las soluciones encontradas para resolver las especificaciones del capítulo 3.

El capítulo 5, "*Descripción funcional. Diagramas*", expone de manera gráfica, a través de diagramas en distintos niveles de descomposición, el funcionamiento del programa. El capítulo 6, "*Fundamentos geométricos*", proporciona los conocimientos matemáticos y/o geométricos en forma de cálculos y ecuaciones que se necesitan en el capítulo 5.

El capítulo 7, "*Implementación*", comenta brevemente cada procedimiento utilizado y su realización en Visual Basic. El capítulo 8, "*Manual del usuario*", muestra cómo se instala, ejecuta y funciona el programa, cuáles son sus cuadros, menús y ventanas, etc.

El capítulo 9, "*Ejemplos*", proporciona un conjunto de ejemplos, suficientemente representativos, para utilizar con el programa. El capítulo 10, "*Conclusiones y desarrollos futuros*", contempla las conclusiones finales del proyecto y los posibles desarrollos futuros y mejoras de la aplicación.

El capítulo 11, "*Bibliografía*", recopila en una lista la documentación empleada, tanto títulos de libros como artículos de revistas y páginas web en Internet. Finalmente, el anexo A, "*Código fuente*", expone la totalidad del código fuente en Visual Basic utilizado por la aplicación.

2
ELIMINACIÓN DE SUPERFICIES OCULTAS

2. ELIMINACIÓN DE SUPERFICIES OCULTAS

El objetivo de este capítulo es mostrar brevemente algunos de los algoritmos de eliminación de superficies ocultas, exponiendo cuál es la idea básica de cada uno y cómo funcionan. Igualmente, se proporciona una posible clasificación de diversos algoritmos relacionados con el problema.

2.1. INTRODUCCIÓN

A mediados de los años 60, se planteó a los informáticos el problema de la eliminación de las líneas y de las superficies ocultas de un sólido opaco. Desde entonces, han aparecido distintos métodos y técnicas para abordar el problema.

Dejando aparte el problema de la eliminación de las líneas ocultas, cuando un modelo debe ser representado con un alto grado de realismo, se necesita trabajar con manchas de color que es lo que hay en la Naturaleza. Salvo los métodos basados en prioridad que suelen operar con polígonos grandes, la mayoría de los algoritmos utilizan técnicas de tipo *puntillista*, es decir, se basan en la idea de que "muchos puntos juntos forman una imagen". O sea, estos algoritmos se concentran en calcular cuál debería ser el color correcto en cada pixel de la pantalla.

2.2. CLASIFICACIÓN DE LOS ALGORITMOS

Según el espacio de trabajo que se utiliza a la hora de analizar la escena se clasifican en:

- a) los que trabajan en el *espacio objeto*.

  Trabajan en un sistema de coordenadas del espacio al que se refiere el objeto o escena a representar. De esta manera, estudian las relaciones entre los objetos de la escena y entre las distintas caras de un objeto con el fin de determinar qué partes de cada uno de los objetos son visibles. Los cálculos son realizados con la máxima precisión posible y las imágenes obtenidas son perfectas sea cual sea la ampliación realizada a la escena. Estos algoritmos se usan en las aplicaciones que necesitan resultados muy precisos. Entre éstos están el algoritmo de Galimberti y Montanari y el de Weiler y Atherton.

- b) los que trabajan en el *espacio imagen*.

  Trabajan en el sistema de coordenadas de la pantalla. Así, estudian la imagen a representar en el monitor teniendo en cuenta el estado final en que debe quedar cada punto de la pantalla o pixel. Los cálculos son realizados con las proyecciones de los objetos sobre el plano imagen con la precisión dependiente de la resolución de la pantalla para que no aparezcan errores en la representación de la

imagen. Por tanto, debido a que la resolución de las pantallas y monitores no es muy alta, el tratamiento de la imagen es más rápido. El principio básico consiste en hallar la luminosidad de cada uno de los pixels. Entre éstos se encuentran el algoritmo de Warnock y el de Watkins.

- c) los que trabajan de forma mixta.

Una primera parte del tratamiento se realiza en el espacio objeto y proporciona una lista de prioridades entre las caras que componen la escena. Posteriormente, la segunda parte del tratamiento de la imagen se realiza en el espacio imagen. Ejemplo de este tipo es el algoritmo de Newell, Newell y Sancha.

A *grosso modo*, podría hacerse una similitud entre las imágenes vectoriales y los métodos del tipo espacio objeto y las imágenes bitmaps y los métodos del tipo espacio imagen. Por supuesto, los métodos mixtos serían similares a las imágenes *metafile*: parte vectorial, parte bitmap.

2.3. ALGORITMOS

Los algoritmos que a continuación se van a exponer son una breve muestra de los distintos métodos que se han desarrollado para resolver el problema de la eliminación de superficies ocultas. En el título de los algoritmos se ha reseñado la bibliografía de la cual se han tomado.

2.3.1. Algoritmo de Galimberti y Montanari [Cor85]

De forma simplificada, el algoritmo que R. Galimberti y U. Montanari crearon en 1979 funciona de la siguiente manera.

Sea un objeto poliédrico, es decir, cualquier conjunto de superficies cerradas y acotadas por caras planas. Los vértices que lo componen se transforman mediante las ecuaciones de proyección y encuadre teniendo en cuenta los lados que forman.

A continuación, el algoritmo examina secuencialmente en el plano cada uno de estos lados para saber qué partes de ellos son visibles. Para ello compara cada uno con todos los demás detectando sus intersecciones. Realizado esto, cualquier lado se tendrá dividido en un número finito de partes donde el punto de unión de cada 2 partes representa una intersección con otro lado.

Se observa que cada una de estas partes son totalmente visibles o totalmente no visibles, ya que todos los puntos que la componen lo son. Por tanto, determinando cómo es un punto cualquiera de cada parte se sabe qué tipo de visibilidad tiene.

Así, el algoritmo declara no visible un punto si se cumple que:

- el punto es interior a un polígono en el plano del dibujo.
- el plano que contiene al polígono en el espacio corta al segmento que une el punto con el observador.

En otro caso, el punto se considera visible y con él, toda la parte entre 2 intersecciones a la que pertenece.

Finalmente, y repitiendo esta prueba para cada parte de un lado, se puede dibujar la parte visible de éste. Asimismo, repitiendo el proceso para cada lado, se tendrá dibujada la parte visible del poliedro.

Hay que hacer constar que este algoritmo tiene su aplicación en poliedros no convexos puesto que en el caso de poliedros convexos hay otros algoritmos más rápidos que éste. No obstante, el algoritmo de Galimberti y Montanari suele usarse como paso previo a la utilización de otros algoritmos y métodos de eliminación de partes ocultas.

2.3.2. Algoritmo de Newell, Newell y Sancha o del pintor [Cor93]

En 1972, M.E. Newell, R.G. Newell y T.L. Sancha desarrollaron un algoritmo de eliminación de partes ocultas basado en la forma en que un pintor realiza sus cuadros, de ahí su nombre.

Básicamente, se trata de lo siguiente. El pintor empieza pintando el fondo. Luego, pinta los objetos del primer plano. No necesita borrar las partes del fondo que van a resultar tapadas, simplemente pinta sobre éstas. La nueva capa de pintura cubrirá a la antigua de modo que sólo la última es la visible.

Por tanto, el núcleo del algoritmo es establecer una ordenación por prioridades entre los objetos y entre los polígonos que los forman, objetivo que no siempre es posible, debiendo de seccionarse en algunos casos un polígono para eliminar indeterminaciones.

Una vez transformada toda la escena por proyección y encuadre y eliminadas las caras posteriores, se realiza una ordenación previa de las caras usando la máxima profundidad de cada una, es decir, el mayor valor de la componente Z de cada vértice de una cara.

Como pueden darse casos de solapamientos entre caras, se emplean otros tests para resolver estas ambigüedades surgidas.

Ya realizada la ordenación, bastará con representar las caras tal y como se han ordenado. Esto es, primero se pintan las más lejanas y después, las más cercanas al observador, resultando visibles las que no han sido tapadas por ninguna otra.

Es de destacar que este algoritmo es excesivamente lento en escenas que tienen muchas caras ya que han de ser comparadas todas entre sí. Además, los casos de indeterminaciones antes citados pueden hacer que se retarde aún más la resolución de una imagen por medio de este algoritmo.

2.3.3. Algoritmo de Warnock [Her96]

El algoritmo de J. Warnock, creado en 1968, también se denomina de subdivisión de áreas.

Se basa principalmente en dividir cada área de la imagen en 4 cuadrados iguales, llamados "ventanas", y averiguar para cada ventana si se está en uno de los casos siguientes:

-1) hay polígonos que contienen totalmente a la ventana

-2) hay polígonos que intersectan con la ventana

-3) hay polígonos contenidos totalmente en la ventana

-4) hay polígonos disjuntos de la ventana

Los polígonos disjuntos no tienen influencia sobre la ventana. Asimismo, la parte de un polígono que intersecta que está fuera de la ventana es irrelevante mientras que la que está dentro puede ser tratada como en el apartado 3).

En los 4 casos puede tomarse una decisión fácil acerca de la ventana, de modo que ésta no necesite dividirse nuevamente:

-1) todos los polígonos son disjuntos de la ventana. La ventana se muestra con el color del fondo.

-2) sólo hay un polígono que intersecta o un polígono contenido. La ventana se pinta primero con el color del fondo y luego, la parte del polígono contenida en la ventana se pinta con su color.

-3) existe un único polígono que contiene la ventana y no existen polígonos contenidos o que intersecten. La ventana se rellena con el color del polígono que la contiene.

-4) existe más de un polígono que intersecta, está contenido o contiene la ventana, pero hay un polígono que contiene la ventana que está delante de los restantes polígono. Para determinar si un polígono que contiene la ventana está por delante, se calculan las coordenadas Z de los planos de todos los polígonos en las esquinas de la ventana. Si hay un polígono que contiene la ventana cuyas coordenadas Z en las esquinas son mayores (más cerca al punto de vista) que las de cualquier otro polígono, entonces la ventana puede rellenarse con el color de este polígono totalmente.

En el caso de ambigüedades, este algoritmo siempre subdivide la ventana para simplificar el problema. Por otra parte, el proceso de subdivisión es de carácter recursivo, quedando claro que el límite del proceso es la propia resolución de la pantalla en la que se trabaja, ya que por muy complicado que sea el contenido de una ventana si el tamaño de ésta es igual a un pixel, es innecesario continuar: simplemente se representa el pixel.

2.3.4. Algoritmo de Watkins [Cor93]

Se trata de un algoritmo creado en 1970 por S. Watkins, quien se basó en el procedimiento de rellenado por barrido usando un y-bucket para preclasificar los lados de los diferentes polígonos.

La idea es la siguiente. Si se intersecta la escena, ya proyectada y encuadrada, por un plano horizontal correspondiente a la ordenada de una línea cualquiera de barrido, producirá una serie de "rodajas" de 2 dimensiones que pueden compararse entre sí, en el plano, para determinar qué parte de ellas es la más cercana al observador y, por tanto, visible.

Como estas secciones planas de la escena son polígonos en el plano horizontal anteriormente citado, se proyectan todos los vértices de estos polígonos sobre el eje de abscisas obteniendo una serie de intervalos, denominados "spans".

El algoritmo clasifica estos spans en 3 categorías:

- no contienen ningún segmento; aparece el color del fondo.

- contiene un único segmento; se muestra el color del único polígono de la escena 3D que ha producido el segmento. (Este caso no puede presentarse si los polígonos se han formado al intersectar poliedros cerrados).
- contienen más de un segmento; hay que averiguar cuál de los segmentos es el más cercano al observador.

Como puede comprobarse, en el interior de un span, nunca se altera la ordenación relativa de los segmentos, de forma que aquel que tiene menor Z en un punto cualquiera, también tiene la menor en todos los demás puntos.

Cuando se determine cuál es el segmento visible, el span se muestra con el color del polígono tridimensional que lo produjo. Al clasificarse todos los spans y dibujarse cada uno con su color apropiado, la línea de barrido de la pantalla tendrá su visibilidad resuelta y podrá pasarse a una nueva línea de barrido.

Este algoritmo es muy rápido y eficiente pero su principal inconveniente es que necesita calcular por cada línea de barrido la intersección del correspondiente plano con toda la escena.

2.3.5. Algoritmo de Weiler y Atherton [Her96]

Basado en el algoritmo de Warnock, el algoritmo desarrollado por K. Weiler y P. Atherton en 1977, subdivide el área de la pantalla a lo largo de los bordes de los polígonos en lugar de a lo largo de fronteras rectangulares. Para ello, se requiere de un potente método de recorte que pueda recortar polígonos cóncavos con agujeros frente a otros.

El primer paso que suele realizarse es ordenar los polígonos por su coordenada Z más cercana al observador. De esta manera, el polígono con menor Z se emplea para recortar todos los polígonos en 2 listas que contienen la parte interior y la parte exterior al polígono de recorte.

Todos los polígonos de la lista interior que estén detrás del polígono de recorte se eliminan, ya que no serán visibles. Si cualquier otro polígono de la lista interior está más cerca que el polígono de recorte, se reordena la lista y se aplica el proceso anterior recursivamente.

A continuación, se analiza la lista exterior de igual modo.

3
DESCRIPCIÓN DEL ALGORITMO Z-BUFFER. ESPECIFICACIONES DEL PROYECTO

3. DESCRIPCIÓN DEL ALGORITMO Z-BUFFER. ESPECIFICACIONES DEL PROYECTO

El objetivo de este capítulo es describir cómo es el algoritmo Z-Buffer, base fundamental de este proyecto. Se verán también cuáles son sus ventajas e inconvenientes.

Además, se muestran las especificaciones recibidas para su desarrollo.

3.1. DESCRIPCIÓN DEL ALGORITMO Z-BUFFER

3.1.1. Introducción

Desarrollado por E. Catmull en 1974, sin embargo, su popularidad no ha sido muy grande hasta la actualidad debido al gran consumo de memoria que utiliza, cuyo coste en la década de los '70 era considerable. Hoy en día, gracias al abaratamiento de las memorias necesarias para tal aplicación y, como no, a su simplicidad, ha crecido su uso hasta el extremo de que la mayoría de las tarjetas gráficas 3D llevan implementado en hardware algún desarrollo del algoritmo Z-Buffer para resolver la eliminación de superficies ocultas a la hora de representar figuras tridimensionales.

3.1.2. Buffers

El algoritmo Z-Buffer consiste principalmente en 2 matrices o buffers de las mismas dimensiones que la resolución de la pantalla donde se va a trabajar con la imagen.

Una de las matrices, denominada *frame-buffer*, almacena en cada posición *(x,y)* un valor que se corresponde con el color que mostrará el pixel de coordenadas *(x,y)* cuando sea dibujado.

La otra matriz, denominada *z-buffer* o buffer de profundidad, es la clave del algoritmo (de ahí su nombre). En ella se guarda la componente *Z* del punto de la escena más cercano al observador que se proyecta sobre cada pixel. Obviamente, se supone que todo lo que aparece en la escena ha sido anteriormente transformado por las ecuaciones de proyección y encuadre correspondientes, por lo que la componente *Z* representa, en realidad, su profundidad.

3.1.3. Algoritmo Z-Buffer [Cor93]

De manera esquematizada, el algoritmo es el siguiente:

1.- Inicialización de la matriz z-buffer, llamada *PROF[x,y]*, a un valor enorme (puede ser *infinito*).

2.- Inicialización de la matriz frame-buffer, llamada *COLOR[x,y]*, al valor del color del fondo de la escena (en la práctica, se suele borrar la pantalla para simular esta inicialización).

3.- Eliminación de las caras posteriores de los objetos de la escena, ya que éstas no son visibles desde el observador, con lo cual se ahorra trabajo al reducirse el número de caras a tratar.

4.- Transformación por proyección y encuadre de cada uno de los vértices de las caras anteriores. Además, dado que todavía no se ha procesado ninguna cara, la matriz *PROF[x,y]* indica en cada pixel que lo más cercano al observador es el fondo de la escena.

5.- Por cada cara transformada, cálculo de todos los pixels que contiene. Suele utilizarse para esta tarea un algoritmo de rellenado de polígonos mediante barrido; eso sí, en este caso no se rellena la cara sino que se averiguan las coordenadas de los distintos pixels que forman parte de la cara.

6.- Por cada uno de estos pixels, cálculo de la profundidad o componente *Z* de la cara, la cual se puede determinar a partir de la ecuación del capítulo 6 apartado 6.6, siendo allí *(XP,YP)* las coordenadas del pixel y *ZP*, la componente *Z*.

6.a.- Luego, comparación de esta profundidad *Z* con la que esté almacenada en *PROF[x,y]*. Si *Z* < *PROF[x,y]*, la cara está más cerca del observador que cualquier otra previamente analizada en ese pixel *(x,y)*. Por tanto, se actualizan ambas matrices resultando: *PROF[x,y] = Z* y *COLOR[x,y] = fc(x,y,z)*, donde *fc(x,y,z)* es una función que indica el color que debe tener la cara tratada en el punto *(x,y,z)*. Por lo general, para que una imagen sea realista se utilizan técnicas de iluminación por lo que el color de las caras no es uniforme, de ahí que se haya empleado una función genérica.

6.b.- Si *Z* ≥ *PROF[x,y],* la cara en el pixel *(x,y)* está más lejos del observador que alguna previamente procesada, por lo que se conservan los valores que ya hay en las matrices.

Una vez terminada la comprobación anterior, se continúa con el siguiente pixel de igual forma y, al finalizar la cara actual, se sigue con la siguiente.

Cuando todo el proceso anterior llegue a su fin, la matriz *COLOR[x,y]* contiene el color correcto que debe mostrarse para cada pixel de la pantalla y el problema de la visibilidad totalmente resuelto.

3.1.4. Ventajas e inconvenientes

Entre las ventajas de este algoritmo cabe destacar la simplicidad de sus cálculos ya que utiliza procedimientos y fórmulas por todos conocidas. Por otra parte, su tiempo de ejecución prácticamente no depende de la complejidad de la escena, sino sólo de su parte visible. Es decir, dos escenas cuyas imágenes finales sean del mismo orden de complejidad serán resueltas aproximadamente en el mismo tiempo, independientemente del número de caras que compongan cada una de las escenas.

En cuanto a sus inconvenientes, el principal es la enorme cantidad de cálculo que requiere puesto que cada pixel es evaluado tantas veces como caras a las que pertenece. Por supuesto, otro de sus fallos, antes citado, es la cantidad de memoria que necesita; sin embargo, la bajada de precios en este ámbito hace del algoritmo Z-Buffer una buena opción.

3.2. ESPECIFICACIONES DEL PROYECTO

3.2.1. Introducción

Para la realización de este proyecto han de tenerse en cuenta ciertas características y especificaciones que se irán desglosando en los siguientes apartados. Estos son:

- demostración del funcionamiento de un algoritmo
- facilidad de uso del programa
- visualización de los datos y los resultados
- flexibilidad en la utilización de los ejemplos

3.2.2. Demostración del funcionamiento de un algoritmo

La idea principal a la hora de elaborar este proyecto es que el usuario pueda ver de manera visual e interactiva cómo funciona un algoritmo concreto de eliminación de superficies ocultas llamado Z-Buffer.

Dicho esto, queda claro que, además de mostrar en pantalla el modelo original y el modelo resultante, tendrá que aparecer el propio algoritmo y visualizar, de alguna manera, cómo se recorre dicho algoritmo y cómo se expone el resultado de las decisiones tomadas en el mismo.

Además, nuevamente debido al carácter didáctico de la aplicación, habrá de permitirse que el usuario pueda interactuar en el desarrollo del algoritmo pudiendo acelerar o detener la ejecución de éste.

3.2.3. Facilidad de uso del programa

El hecho de ser una aplicación de apoyo hace que, de alguna manera, el usuario conozca el objetivo de la misma. Sin embargo, su utilización no tiene porqué ser completamente intuitiva. Es decir, no se pretende que el usuario de programas informáticos conozca el funcionamiento de *todos* los que existen. No obstante, éste que se presenta aquí cumple con unos requisitos básicos para que su aprendizaje sea muy fácil.

Entre esos requisitos, el principal es el empleo del ratón como forma de comunicarse con el ordenador. Como en la gran mayoría de las aplicaciones actuales, disponer del ratón a la hora de trabajar da una cierta movilidad frente al ordenador que no se tiene con el teclado. En este caso, imagínese la utilización del ordenador acompañando a una explicación en una pizarra y podrá comprenderse si es o no más útil el ratón que el teclado.

Otro requisito es, como se verá a continuación, disponer de zonas para visualizar los datos y los resultados, tanto numéricos como gráficos, sin que éstos se solapen o impidan que se pueda acceder a la información en un momento dado.

3.2.4. Visualización de los datos y los resultados

Al ser una aplicación de carácter gráfico, tanto en su objetivo como en su concepto, la visualización de los datos y los resultados habrá de ser lo más clara posible. Para ello, se debe dividir la pantalla en tantas zonas de trabajo como sean necesarias.

Para empezar, es conveniente que los datos originales y los resultados, ambos de carácter gráfico al ser imágenes, ocupen zonas determinadas y exclusivas de la pantalla. El motivo es que el usuario sepa de dónde parte y adónde quiere llegar. Por otra parte, también es necesario que se disponga, de forma visible, el propio algoritmo Z-Buffer.

Finalmente, para facilitar la comprobación del funcionamiento del algoritmo se deben implementar zonas que muestren los datos numéricos que se han ido obteniendo durante la ejecución del algoritmo.

3.2.5. Flexibilidad en la utilización de los ejemplos

Para terminar con las especificaciones de este proyecto, se pide que los ejemplos puedan modificarse con cierta comodidad, ya sea de manera interactiva o preparando los modelos previamente a la ejecución del programa.

Permitir que el usuario tenga cierto control sobre los datos de ejemplos implica, en cierta medida, que sea el mismo quien cree sus propios modelos. Para ello, el uso de ficheros con formato propio es una buena solución, ya que, basta con aprender cómo es la estructura de los ficheros y construir otros ejemplos basados en dicha estructura con los datos correspondientes.

Igualmente, se piensa que, durante la ejecución del programa, se debe permitir al usuario la posibilidad de cambiar algunos datos referentes a los modelos como pueden ser la posición del observador o el punto de mira de éste.

4
SOLUCIONES ADOPTADAS DEL PROYECTO

4. SOLUCIONES ADOPTADAS DEL PROYECTO

El objetivo de este capítulo es cubrir las soluciones encontradas para resolver las especificaciones del capítulo anterior. Se tratan los cuadros y menús, aparte de la presentación y los modelos que se emplearán en la aplicación.

4.1. PRESENTACIÓN

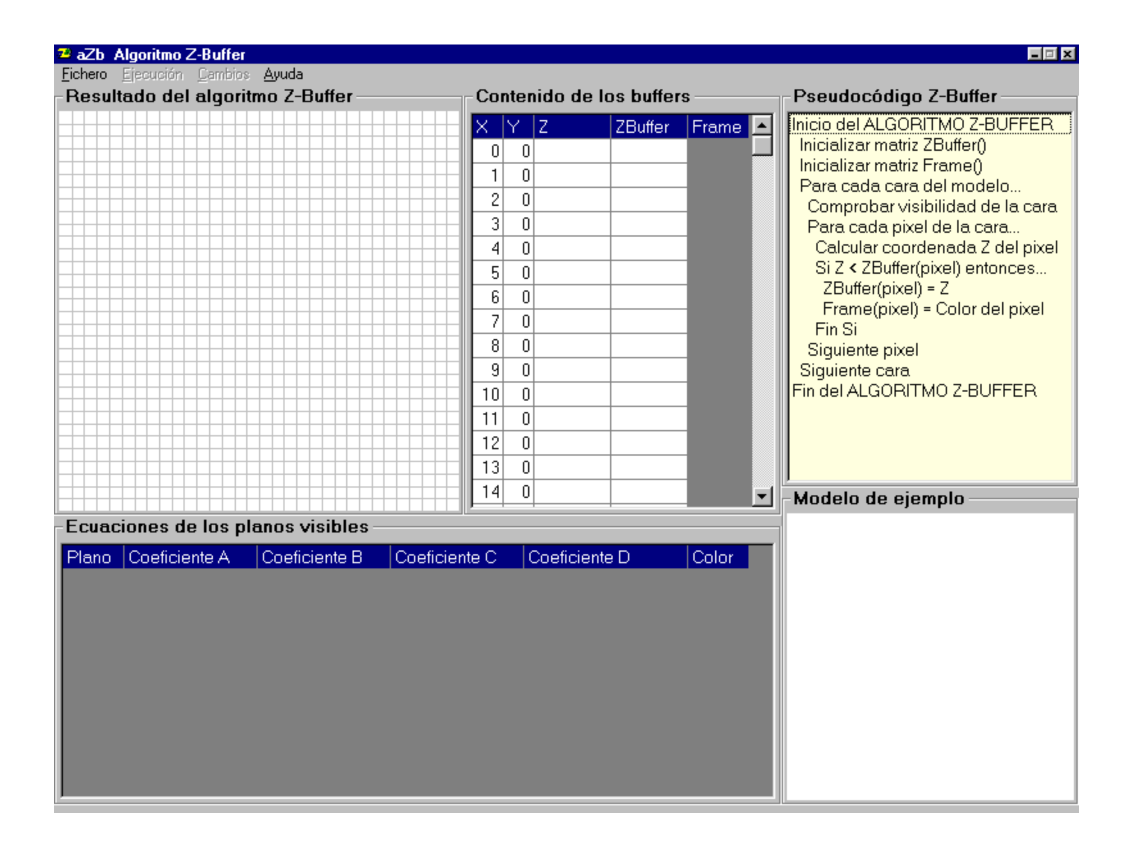
Partiendo del carácter didáctico de esta aplicación, la consideración de que el usuario debería disponer de la totalidad de la pantalla, fuese cual fuese la resolución de la misma, era vital. Esto es, se eliminó la posibilidad de que tener que manejar muchas ventanas y que éstas puedan solaparse entorpeciendo la visibilidad tanto de los modelos originales como de los cálculos y resultados obtenidos.

Para trabajar adecuadamente se divide la ventana principal en 5 partes, que llamaremos *cuadros*, con un cometido particular cada uno:

- *Resultado del algoritmo Z-Buffer*, donde se obtiene el modelo resultante.
- *Contenido de los buffers*, donde se puede observar los valores que hay en cada buffer.
- *Pseudocódigo Z-Buffer*, donde el usuario puede seguir cómo se ejecuta el algoritmo Z-Buffer.
- *Ecuaciones de los planos visibles*, donde se muestran los coeficientes de estas ecuaciones.
- *Modelo de ejemplo*, donde se representa el modelo original.

Asimismo, constará de 4 menús con distintas opciones cada uno (2 de ellos no disponibles al principio):

- *Fichero*
- *Ejecución*
- *Cambios*
- *Ayuda*



4.2. CUADROS

4.2.1. Cuadro Resultado del algoritmo Z-Buffer

En este cuadro el usuario va obteniendo el resultado de la aplicación del algoritmo Z-Buffer sobre el modelo de ejemplo que disponga. No se tratará de mostrar el resultado completo final sino, más bien, de mostrar cómo se va logrando paso a paso dicho resultado.

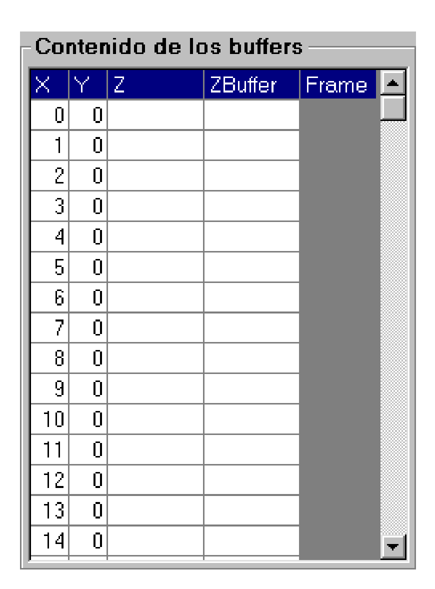


Para ello, y por problemas de resolución y tiempo, se adopta un cuadro de trabajo dividido en 32x32 casillas, cada una de las cuales ocuparán varios pixels de la pantalla. El lector podrá entonces comprender que esta opción tiene ciertos inconvenientes, siendo el principal que el resultado es una muestra *grotesca* del modelo con pixels muy *grandes* (los llamaremos *zbpixels*). Por otra parte, las aristas sufren el efecto del *aliasing* en gran medida. Y, además, se encuentra que, debido a la bajísima resolución con la que se trabaja, ciertas partes de los modelos no se dibujan con la exactitud necesaria.

Sin embargo, trabajar con un cuadro tan pequeño también tiene sus ventajas. Entre ellas está la reducción del tiempo de procesado ya que éste es proporcional al tamaño del cuadro de trabajo y aquí se ha reducido. Al tener un tamaño de 32x32 (1.024 casillas) frente al usual en modelado de 800x600 (480.000 casillas), se reduce el tiempo de procesado de manera *drástica*, con lo cual el tiempo de exposición es menor. Además, el usuario puede controlar con mayor facilidad en todo momento dónde está siendo aplicado el algoritmo, ya que se cuenta con un número reducido de puntos de dibujo, los *zbpixels*.

4.2.2. *Cuadro* Contenido de los buffers

En este cuadro se visualizan los datos que se van obteniendo y que llenan los dos buffers: el *z-buffer* o buffer de profundidades y el *frame-buffer* o buffer de colores.



La cuadrícula tendrá 5 columnas por 1025 filas. En las dos primeras columnas se indicarán la coordenada X y la coordenada Y de la casilla en la que se está aplicando el algoritmo en un momento dado. En las otras tres tendremos la coordenada Z, el contenido del *z-buffer* y el contenido del *frame-buffer*. Las 1025 filas serán la cabecera de la cuadrícula y una fila por cada *zbpixel* del cuadro *Resultado del algoritmo Z-Buffer*.

4.2.3. *Cuadro* Pseudocódigo Z-Buffer

El pseudocódigo del algoritmo Z-Buffer se expondrá en este cuadro para que el usuario pueda seguir de manera interactiva cómo se va resolviendo el problema de la visibilidad en el modelo de ejemplo.

Pseudocódigo Z-Buffer

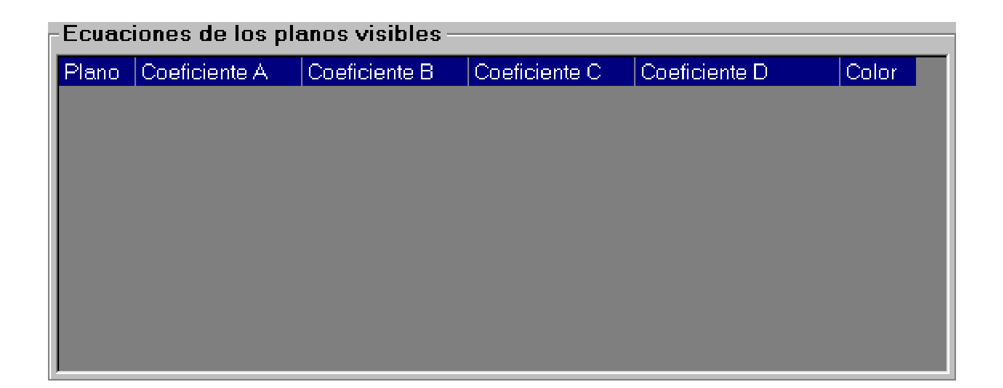
```
Inicio del ALGORITMO Z-BUFFER
 Inicializar matriz ZBuffer()
 Inicializar matriz Frame()
 Para cada cara del modelo...
  Comprobar visibilidad de la cara
  Para cada pixel de la cara...
   Calcular coordenada Z del pixel
   Si Z < ZBuffer(pixel) entonces...
    ZBuffer(pixel) = Z
    Frame(pixel) = Color del pixel
   Fin Si
  Siguiente pixel
 Siguiente cara
Fin del ALGORITMO Z-BUFFER
```

Dependiendo del modo de ejecución, automática o manual, se ve cómo la línea correspondiente del algoritmo Z-Buffer se va aplicando en ese preciso momento. En el caso de la ejecución automática, se verá una barra de color que se mueve de una línea a otra automáticamente. Sin embargo, en los dos bucles *Para...Siguiente* y en la condición *Si...entonces*, la barra de color azul se moverá hasta el lugar que le corresponda. Si la ejecución es manual, el usuario podrá moverse a través del pseudocódigo.

Por otra parte, indicar que el algoritmo Z-Buffer escrito en este cuadro se adaptará y simplificará adecuadamente para poder presentarlo en pocas líneas y, de esta manera, seguir fácilmente el recorrido de la barra a lo largo de todo el pseudocódigo.

4.2.4. Cuadro Ecuaciones de los planos visibles

Este cuadro es similar al del *Contenido de los buffers*, ya que se trata también de una cuadrícula. Constará de 6 columnas y un número arbitrario de filas, según cada modelo de ejemplo, por lo que al iniciarse la aplicación sólo se mostrará la cabecera.



Las 6 columnas se corresponderán con el nombre del plano, los coeficientes A, B, C y D de la ecuación del plano $(A \cdot X+B \cdot Y+C \cdot Z+D=0)$ y el color de dicho plano. En cuanto a las filas, éstas irán apareciendo conforme vayan surgiendo de la ejecución del algoritmo Z-Buffer. Además, cuando se cargue un nuevo modelo o se realice algún cambio en las coordenadas, la cuadrícula se borrará completamente excepto la cabecera que siempre se mantendrá fija.

4.2.5. Cuadro Modelo de ejemplo

Este cuadro tendrá unas dimensiones de 320x320 y es el último de la ventana principal. En él se visualizará el modelo de ejemplo como modelo de alambre, es decir, con todas las aristas visibles y todas las caras sin color y transparentes.

Modelo de ejemplo

Debido a este modelo de alambre se observará lo complejo que puede llegar a ser el discernir qué caras son visibles y cuáles no lo son, hecho que facilitará el porqué de una aplicación como la que aquí se expone.

También se va a considerar siempre un encuadre de la proyección del modelo tal que no exista el problema de *clipping* de las líneas que componen dicho modelo de alambre. Como consecuencia, siempre se verá un modelo ajustado a los límites del cuadro pero sin llegar a tocar éstos debido a un *margen de seguridad* adecuado por la propia aplicación.

Finalmente, explicar que a la vez que se va recorriendo el pseudocódigo del algoritmo Z-Buffer, en este modelo de ejemplo se irá coloreando la cara actual para resaltarla del resto de las caras del modelo.

4.3. MENÚS

4.3.1. Menú Fichero

En este primer menú se dispone de las opciones básicas en cualquier aplicación de *Abrir...* y *Salir*.

Mediante la opción *Abrir...* se tiene acceso a los distintos ficheros con extensión *.aZb*, que se corresponderán con los que tienen la estructura adecuada para poder ser leidos por este programa y que contendrán un modelo de ejemplo.

La otra opción, *Salir*, permitirá salir correctamente del programa

4.3.2. Menú Ejecución

Este menú no estará disponible hasta que no exista un modelo en la memoria del ordenador, o sea, hasta que no se haya cargado un fichero que contenga un modelo de ejemplo. Cuando ya pueda utilizarlo, se dispondrá de dos opciones: *Automática* y *Manual*.

La opción *Automática* permitirá la ejecución del algoritmo Z-Buffer de manera automática, sin intervención por parte del usuario, salvo la posibilidad de realizar pausas.

La segunda opción, *Manual*, dejará que sea el usuario el que lleve el ritmo de ejecución del algoritmo puesto que ha de ser éste el que haga avanzar la barra a través del pseudocódigo.

Hay que destacar dos cosas más. Primero, que cuando se termine de cargar un modelo de ejemplo o se cambien las coordenadas del modelo, aparecerá una ventana que pregunte cómo va a ser la ejecución; por tanto, este menú no se usará. Segundo, que se puede pasar de una ejecución manual a una automática, con el objeto de acelerar el recorrido por el pseudocódigo, y viceversa.

4.3.3. Menú Cambios

El tercer menú de la ventana principal tampoco estará disponible al principio y su aparición será igual a la del menú *Ejecución*. Constará de tres opciones similares entre sí, a saber: *Observador...*, *Punto de mira...* y *Dirección de mira...*.

Al elegir una de ellas, se mostrará una ventana que permita modificar las coordenadas del observador, las coordenadas del punto de mira o las componentes de la dirección de mira, según la opción escogida.

Si se confirma el cambio en las coordenadas, la ejecución, ya sea automática o manual, se detendrá, se redibujará el modelo de alambre a su nuevo estado y se preguntará cómo va a ser la ejecución.

4.3.4. Menú Ayuda

Finalmente, en este menú el usuario dispondrá de dos opciones que son: *Contenido* y *Autor*.

Eligiendo la opción *Contenido*, se visualizará la ventana de *Ayuda*, la cual mostrará el contenido del fichero de ayuda en formato HTML llamado *aZb.htm.*

Por último, mediante la opción *Autor* se podrá ver la ventana inicial que aparece al ejecutarse la aplicación y que indicará cuál es su título y quién es su autor, es decir, los créditos del programa.

4.4. MODELOS

Un detalle añadido al aspecto pedagógico de esta herramienta es el tipo de modelo que se va a emplear como ejemplos.

Entre las distintas posibilidades que se tenían, se ha optado por la utilización de ficheros con un único modelo de tipo poliédrico, cóncavo o convexo, de caras planas y coloreadas con colores únicos básicos. De esta manera, se descartan posibles escenas con varios modelos, la aparición de modelos con superficies de revolución o caras alabeadas, el uso de texturas o caras con varios colores y la disposición de grandes paletas de colores.

Resumiendo, en cada fichero aparecerán las 3 coordenadas *(X,Y,Z)* de cada uno de los vértices que componen el modelo, seguidas de unas listas de declaración de los vértices que forman cada una de las caras, así como el color que tiene cada una de ellas. Finalmente, aparecerán las coordenadas *(X,Y,Z)* de posición del observador y del punto de mira. Cualquier defecto en la estructura del fichero deberá ser detectado al leerse por la aplicación, no obstante, ésta será una detección sintáctica, no geométrica.

5
DESCRIPCIÓN FUNCIONAL. DIAGRAMAS

5. DESCRIPCIÓN FUNCIONAL. DIAGRAMAS

El objetivo de este capítulo es exponer gráficamente, a través de diagramas en distintos niveles de descomposición, el funcionamiento del programa. Además, el lector puede hacerse una idea global de la aplicación desde el punto de vista funcional.

5.1. NIVEL 1

Como en la gran mayoría de los programas informáticos, en un primer nivel se tienen 3 bloques:

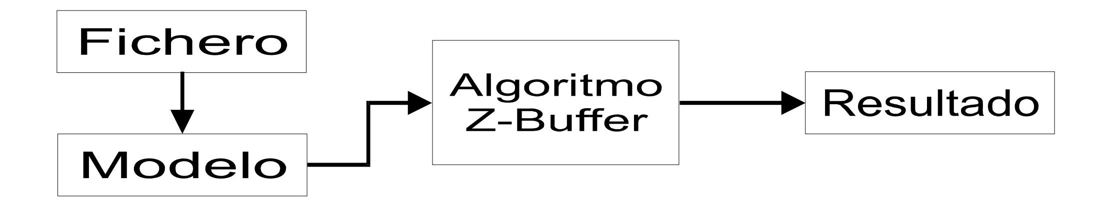


- *Entrada*: Facilita los datos necesarios a *Proceso* para que el programa realice su tarea.
- *Proceso*: Recibe los datos de *Entrada* y genera los resultados de *Salida*, siendo la parte principal de la aplicación.
- *Salida*: Muestra los resultados producidos en *Proceso*.

5.2. NIVEL 2

Dividiendo los bloques del nivel 1, el bloque *Entrada* queda como:

- *Fichero*: Contiene los datos correspondientes al modelo de ejemplo, al observador y al punto de mira de la escena a representar.
- *Modelo*: Se encarga de mostrar una vista preliminar del ejemplo obtenido del fichero.



En cuanto al bloque *Proceso*, se compone únicamente de:

- *Algoritmo Z-Buffer*: Lleva a cabo la ejecución del propio algoritmo, fundamento del programa.

Finalmente, el bloque *Salida* tiene:

- *Resultado*: Traza el dibujo final como consecuencia del algoritmo.

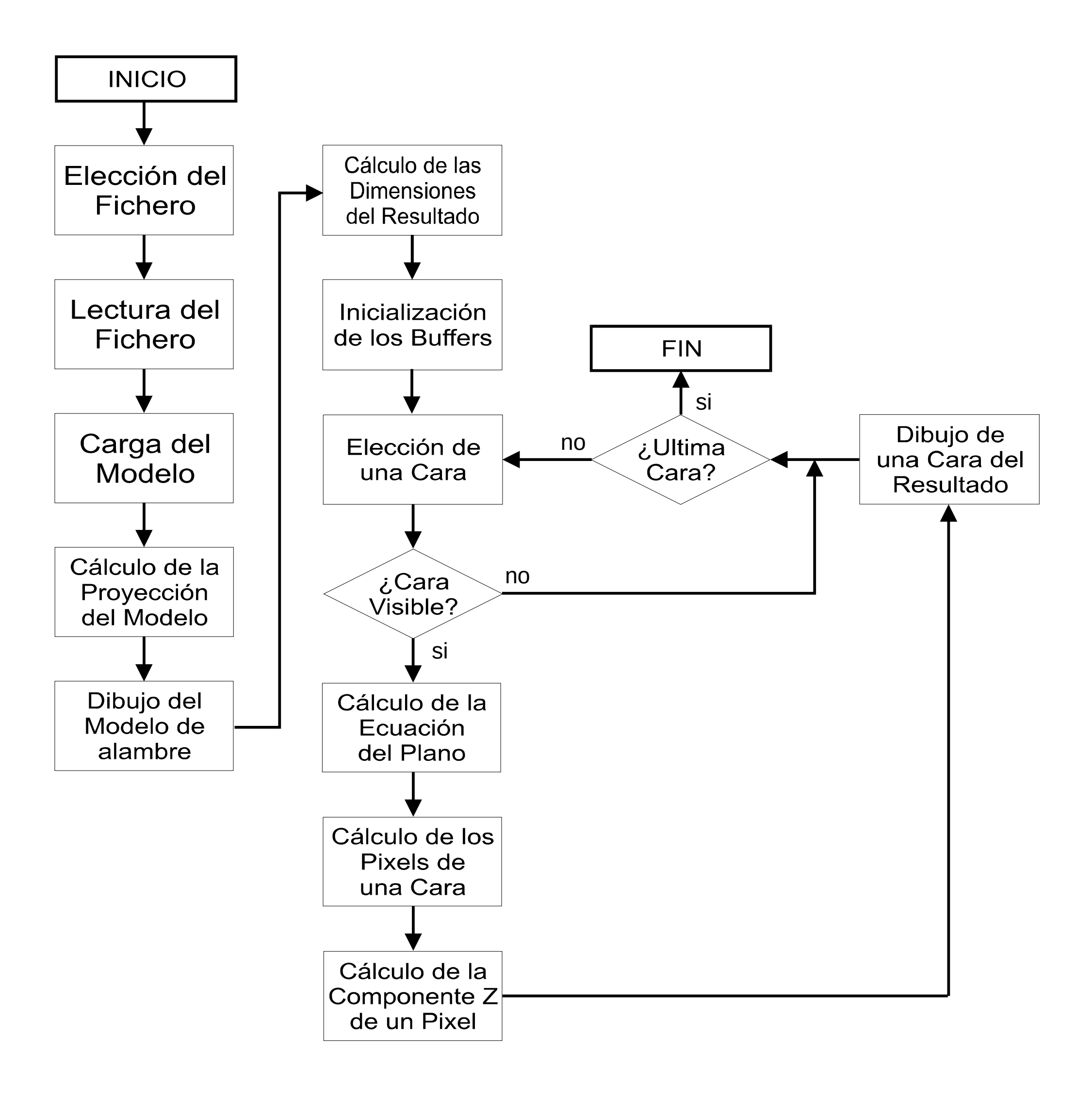
5.3. NIVEL 3

Una vez descompuestos los bloques del nivel 2, el bloque *Fichero* comprende:

- *Elección del Fichero*: El usuario puede elegir el fichero *.aZb* que contiene los datos del modelo en un formato particular.
- *Lectura del Fichero*: Tras elegir el fichero, el programa lee los datos del modelo y los interpreta para usarlos correctamente.

Asimismo, el bloque *Modelo* se divide en:

- *Carga del Modelo*: Después de leer el fichero, la aplicación dispone ya de todo lo necesario para representar el modelo adecuadamente.
- *Cálculo de la Proyección del Modelo*: Con los datos obtenidos y haciendo uso de las expresiones correspondientes (ver capítulo 6, apartado 1), se determina la conversión del espacio 3D al plano 2D.
- *Dibujo del Modelo de alambre*: Una vez realizado el cálculo, el programa dibuja el modelo mostrando solo sus vértices y sus aristas, sin pintar las caras.



En el bloque *Algoritmo Z-Buffer* se encuentran las partes principales del programa:

- *Cálculo de las Dimensiones del Resultado*: Para poder representar de manera adecuada el resultado, la aplicación calcula cuáles deben ser las dimensiones del cuadro que lo contendrá (ver capítulo 6, apartado 2).

- *Inicialización de los Buffers*: Aquí comienza, en realidad, el algoritmo Z-Buffer y para ello, se inicializan las matrices o buffers a sus valores correspondientes.
- *Elección de una Cara*: A continuación, se elige una cara del modelo de ejemplo (a la que se llama "cara actual") con la cual trabaja el programa.
- *¿Cara Visible?*: Se comprueba la visibilidad de la cara actual. En caso de que no sea visible, se toma una nueva cara.
- *Cálculo de la Ecuación del Plano*: A partir de los vértices de la cara actual, se determina su ecuación que permite, más adelante, conocer la componente *Z* de un pixel (ver capítulo 6, apartado 3).
- *Cálculo de los Pixels de una Cara*: El programa crea una "región" con los pixels (en realidad, *zbpixels*) que pertenecen a la cara actual (ver capítulo 6, apartado 4).
- *Cálculo de la Componente Z de un Pixel*: Después, se halla la componente *Z* o profundidad de cada uno de los zbpixels de la cara actual (ver capítulo 6, apartado 5).

- *Dibujo de una Cara del Resultado*: Luego, se dibuja la cara actual con su color correspondiente. (Aunque éste es el único bloque tras descomponer *Resultado* del nivel 2, se ha expuesto aquí para dar cierta continuidad a la descripción del diagrama anterior).
- *¿Ultima Cara?*: Cerrando el bucle y para terminar, se pregunta si la cara actual es o no la última del modelo. En caso negativo, se elige una nueva cara. En otro caso, se llega al fin del proceso obteniéndose el modelo resultante con la eliminación de sus superficies ocultas.

6
FUNDAMENTOS GEOMÉTRICOS

6. FUNDAMENTOS GEOMÉTRICOS

El objetivo de este capítulo es proporcionar al lector los conocimientos matemáticos y/o geométricos en forma de cálculos, fórmulas y ecuaciones que se necesitan en el capítulo anterior para la resolución de la implementación del algoritmo Z-Buffer.

Para facilitar la localización donde se utilizan estas ecuaciones, los apartados van titulados como los bloques del capítulo 5 en los cuales se ha hablado de ellos.

6.1. CÁLCULO DE LA PROYECCIÓN DEL MODELO

Según *[Don86]*, primero se realiza una traslación del sistema de coordenadas del observador *Obs(XO,YO,ZO)* al sistema de coordenadas del punto de mira *Mir(XM,YM,ZM)*, obteniéndose la dirección de mira *Dir(XD,YD,ZD).*

$$XD = XO - XM$$

$$YD = YO - YM$$

$$ZD = ZO - ZM$$

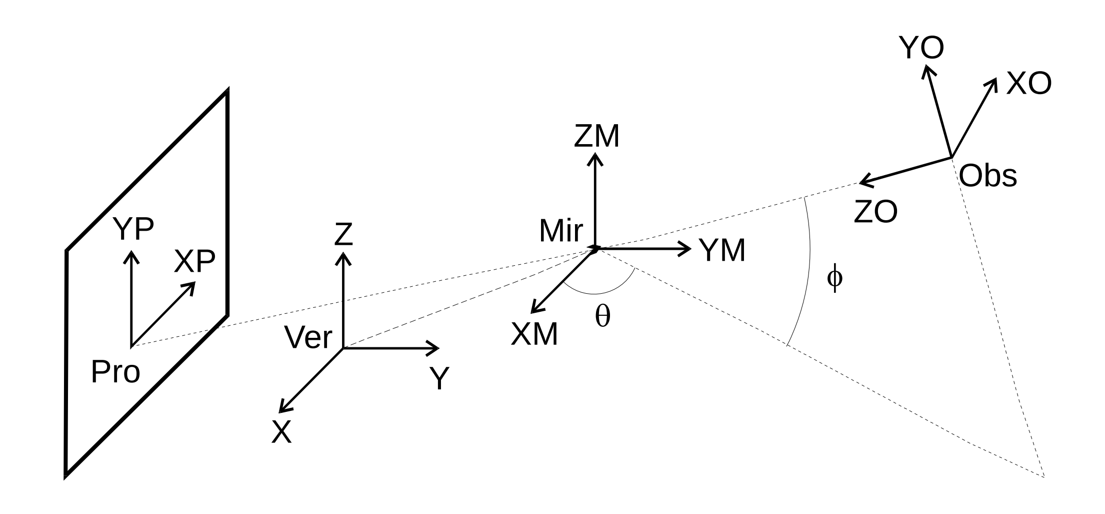
Luego, se convierten dichas componentes de un sistema de coordenadas cartesianas a un sistema de coordenadas esféricas transformándose en *Dir(ρ,θ,ϕ).*

$$\rho = \sqrt{XD^2 + YD^2 + ZD^2}$$

$$\theta = arctg\left(\frac{YD}{XD}\right)$$

$$\phi = arctg\left(\frac{ZD}{\sqrt{XD^2 + YD^2}}\right)$$

A continuación, y aplicando los cambios necesarios según la siguiente figura para transformar los puntos *Ver(X,Y,Z)* al sistema de coordenadas del observador *Obs(XO,YO,ZO)*, se tienen las coordenadas en proyección *Pro(XP,YP,ZP)*.



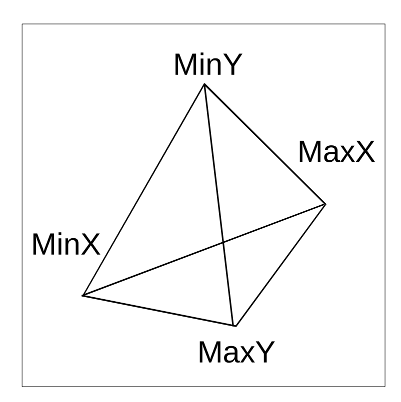
$$XP = -sen\theta \cdot (X - XM) + cos\theta \cdot (Y - YM)$$

$$YP = -cos\theta \cdot sen\phi \cdot (X - XM) - sen\theta \cdot sen\phi \cdot (Y - YM) + \\ + cos\phi \cdot (Z - ZM)$$

$$ZP = -cos\theta \cdot cos\phi \cdot (X - XM) - sen\theta \cdot cos\phi \cdot (Y - YM) - \\ - sen\phi \cdot (Z - ZM) + \rho$$

6.2. CÁLCULO DE LAS DIMENSIONES DEL RESULTADO

Para comenzar, se hallan los valores mínimos y máximos (extremos) de todos los vértices una vez proyectados, siendo *MinX, MaxX, MinY* y *MaxY*.



Entonces, se calculan las diferencias de los extremos de *X* (*DX*) y de *Y* (*DY*) y se toma la mayor de ellas, *Dim*, obteniéndose la dimensión del cuadrado máximo donde representar el modelo.

```
DX = MaxX - MinX
DY = MaxY - MinY
Dim = máx(DX,DY)
```

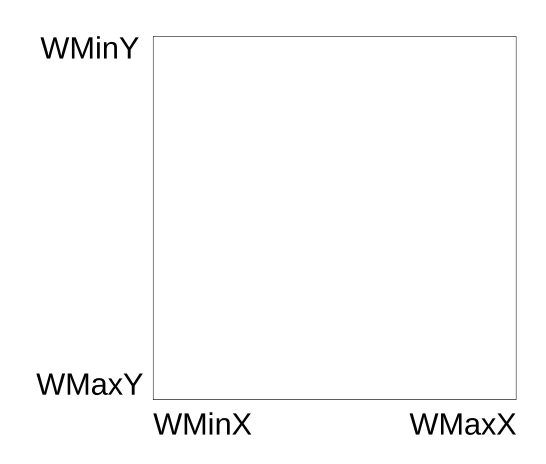
A continuación, haciendo uso de los valores anteriores, se calculan los extremos del "window" o ventana *WMinX, WMaxX, WMinY* y *WMaxY*.

$$\mathrm{WMinX} = \mathrm{MinX} - \left(\frac{\mathrm{Dim} - \mathrm{DX}}{2}\right)$$

$$\mathrm{WMaxX} = \mathrm{MaxX} + \left(\frac{\mathrm{Dim} - \mathrm{DX}}{2}\right)$$

$$\mathrm{WMinY} = \mathrm{MinY} - \left(\frac{\mathrm{Dim} - \mathrm{DY}}{2}\right)$$

$$\mathrm{WMaxY} = \mathrm{MaxY} + \left(\frac{\mathrm{Dim} - \mathrm{DY}}{2}\right)$$



Para que la imagen resultante quede siempre centrada y no haya necesidad de usar algún algoritmo de "clipping", se establece un margen de seguridad *Marg* y con él se calculan los extremos del "viewport" o puerta *VMinX, VMaxX, VMinY* y *VMaxY*.

$$\text{Marg} = 0.05 \cdot \text{Tam}$$

$$\text{VMinX} = \text{Marg}$$

$$\text{VMaxX} = \text{Tam} - \text{Marg}$$

$$\text{VMinY} = \text{VMinX}$$

$$\text{VMaxY} = \text{VMaxX}$$

Tam es el tamaño máximo del cuadro donde se va a mostrar el modelo. Así, para el cuadro *Modelo de ejemplo*, *Tam* vale 319; mientras que para el cuadro *Resultado del algoritmo Z-Buffer* vale 31.

Finalmente, se calculan las coordenadas del encuadre *Enc(XE,YE)* correspondientes a cada vértice *Pro(XP,YP,ZP)* anteriormente hallado.

$$\text{XE} = \frac{(\text{VMaxX} - \text{VMinX}) \cdot \text{XP}}{\text{WMaxX} - \text{WMinX}} + \frac{(\text{VMinX} \cdot \text{WMaxX} - \text{VMaxX} \cdot \text{WMinX})}{\text{WMaxX} - \text{WMinX}}$$

$$\text{YE} = \text{Tam} - \left(\frac{(\text{VMaxY} - \text{VMinY}) \cdot \text{YP}}{\text{WMaxY} - \text{WMinY}} + \frac{(\text{VMinY} \cdot \text{WMaxY} - \text{VMaxY} \cdot \text{WMinY})}{\text{WMaxY} - \text{WMinY}} \right)$$

6.3. CÁLCULO DE LA ECUACIÓN DEL PLANO

Según el método de Newell, descrito en *[Bur89]*, para hallar la visibilidad de una cara se calcula el coeficiente *C* de su vector normal *(A,B,C)* o, lo que es lo mismo, su ecuación del plano *(A·X+B·Y+C·Z+D=0),* a partir de los *n* vértices *Pro(XP,YP,ZP)* de dicha cara.c

$$C = (XP_n - XP_1) \cdot (YP_n + YP_1) + \\ + \sum_{i=1}^{n-1} (XP_i - XP_{i+1}) \cdot (YP_i + YP_{i+1})$$

Si el coeficiente *C* es mayor o igual a 0 (*C≥0)*, la cara será visible. Entonces, se determinan los coeficientes *A* y *B*.

$$A = (YP_n - YP_1) \cdot (ZP_n + ZP_1) + \\ + \sum_{i=1}^{n-1} (YP_i - YP_{i+1}) \cdot (ZP_i + ZP_{i+1})$$

$$B = (ZP_n - ZP_1) \cdot (XP_n + XP_1) + \\ + \sum_{i=1}^{n-1} (ZP_i - ZP_{i+1}) \cdot (XP_i + XP_{i+1})$$

Para terminar, se halla el coeficiente D despejando en la ecuación inicial y sustituyendo uno de los vértices *Pro(XP,YP,ZP)* en ella.

$$D = -(A \cdot XP + B \cdot YP + C \cdot ZP)$$

6.4. CÁLCULO DE LOS PIXELS DE UNA CARA

Se trazan las aristas entre cada vértice y su siguiente utilizando la ecuación de la recta que pasa por dichos vértices. De esta manera, y para cada línea de barrido *Y* desde la *Y* mínima hasta la *Y* máxima que intersecta la arista, se obtiene la correspondiente coordenada *X* del zbpixel intersección.

$\forall\ i=1..n-1$

$$DX = XE_{i+1} - XE_i$$

$$DY = YE_{i+1} - YE_i$$

$$X = XE_i + (Y - YE_i) \cdot \frac{DX}{DY}$$

$\forall\ i=n$

$$DX = XE_1 - XE_n$$

$$DY = YE_1 - YE_n$$

$$X = XE_n + (Y - YE_n) \cdot \frac{DX}{DY}$$

Para el interior de la cara, se emplean las mismas líneas de barrido anteriores y se van uniendo dos a dos los zbpixels que estén en la misma línea de barrido.

6.5. CÁLCULO DE LA COMPONENTE Z DE UN PIXEL

Una vez resuelta la ecuación del plano, se calcula el valor de la componente *Z* con los coeficientes de dicha ecuación.

$$ZP = -\frac{A \cdot XP + B \cdot YP + D}{C}$$

7
IMPLEMENTACIÓN

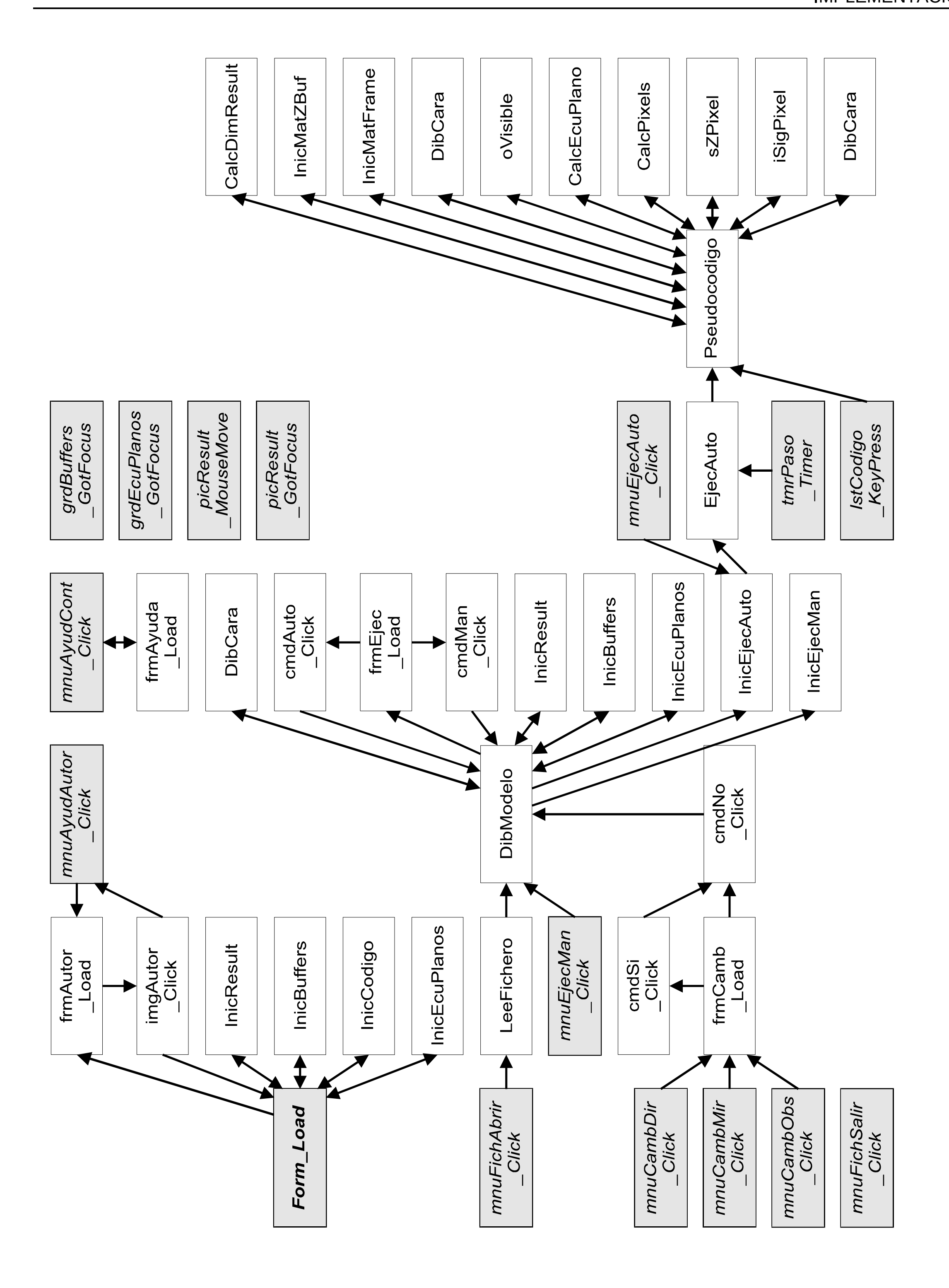
7. IMPLEMENTACIÓN

El objetivo de este capítulo es comentar brevemente cada procedimiento utilizado y su realización en Visual Basic. Eso sí, no se trata todavía del código fuente ya que éste aparece en el anexo A.

7.1. INTRODUCCIÓN

A la hora de implementar el algoritmo Z-Buffer como herramienta didáctica se pensó que lo más adecuado sería utilizar un entorno gráfico, ampliamente conocido y, sobre todo, fácil de usar. En la actualidad, existen tres sistemas operativos que cumplen dichas características: *Apple MacOS*, *Unix/Linux X-Window* y *Microsoft Windows* (hay otros, como *IBM OS/2 Warp*, pero son de uso minoritario). Sin embargo, mientras el primero necesita un ordenador no compatible con la mayoría y el segundo aún no cuenta con un amplio número de usuarios, el último de los expuestos es, a nuestro entender, la mejor opción. Si a sus cualidades antes citadas, añadimos la posibilidad de utilizar un entorno de programación visual como es *Visual Basic 5.0*, última versión hasta la fecha, veremos que el resultado es una aplicación de carácter profesional muy "amigable" con el usuario.

Por otra parte, a la hora de exponer cada una de las funciones y subrutinas (procedimientos), podría seguirse el orden de ejecución del programa. Pero, al estar relacionadas entre sí y ser una aplicación activada por eventos, es preferible explicarlas en orden alfabético (como se encuentra en los listados del anexo A). El diagrama siguiente puede servir de referencia para establecer las distintas relaciones.


CalcDimResult
InicMatZBuf
InicMatFrame
DibCara
oVisible
CalcEcuPlano
CalcPixels
sZPixel
iSigPixel
DibCara
Pseudocodigo
grdBuffers _GotFocus
grdEcuPlanos _GotFocus
picResult _MouseMove
picResult _GotFocus
mnuEjecAuto _Click
EjecAuto
tmrPaso _Timer
lstCodigo _KeyPress
mnuAyudCont _Click
frmAyuda _Load
DibCara
cmdAuto _Click
frmEjec _Load
cmdMan _Click
InicResult
InicBuffers
InicEcuPlanos
InicEjecAuto
InicEjecMan
mnuAyudAutor _Click
DibModelo
cmdNo _Click
frmAutor _Load
imgAutor _Click
InicResult
InicBuffers
InicCodigo
InicEcuPlanos
LeeFichero
mnuEjecMan _Click
cmdSi _Click
frmCamb _Load
Form_Load
mnuFichAbrir _Click
mnuCambDir _Click
mnuCambMir _Click
mnuCambObs _Click
mnuFichSalir _Click

7.2. FORM FRMAZB

7.2.1. Sub CalcDimResult()

Calcula las dimensiones del "viewport" y las coordenadas del cuadro *Resultado del algoritmo Z-Buffer*. Para cada vértice del modelo de ejemplo, determina las coordenadas *X* e *Y*, que se corresponden con las coordenadas transformadas por proyección y encuadre y que se representan en el cuadro *Resultado del algoritmo Z-Buffer*.

7.2.2. Sub CalcEcuPlano(bCara As Byte)

Calcula la ecuación del plano de la cara *bCara*, según el método de Newell. Para cada vértice de la cara *bCara*, determina los coeficientes *A* y *B* de la ecuación del plano.

Luego, toma el coeficiente *C*, previamente hallado en la función *oVisible* y junto con los coeficientes anteriores, calcula el coeficiente *D* (ver capítulo 6, apartado 3). Finalmente, rellena el cuadro *Ecuaciones de los planos visibles* con los datos obtenidos.

7.2.3. Sub CalcPixels(bCara As Byte)

Calcula los pixels correspondientes a la cara *bCara*. Primero, inicializa la matriz de pixels de la cara *bCara*. Para cada vértice de la cara *bCara*, traza la arista con su vértice siguiente. Asimismo, determina las *Y* mínima y máxima para saber cuáles son el primer y último pixel de la cara *bCara*.

A continuación, para cada línea de barrido, rellena la cara *bCara* comprobando si esta línea coincide con los vértices (en ese caso, desplaza "algo" hacia arriba la línea de barrido).

Además, para cada vértice de la cara *bCara*, se van hallando los distintos elementos de la línea de barrido que, luego, ordena de menor a mayor para poder completar la matriz de pixels de la cara *bCara*.

En ciertos casos que pudieran surgir duplicidad de pixels (dos elementos iguales en una misma línea de barrido), se desplaza dicho elemento al siguiente.

7.2.4. Sub DibCara(bCara As Byte, bColor As Byte)

Dibuja la cara *bCara* del modelo de color *bColor* en el cuadro *Modelo de ejemplo*. Para cada vértice de la cara *bCara*, traza la arista con el vértice siguiente construyendo así el modelo de alambre.

7.2.5. Sub DibModelo()

Dibuja el modelo mediante una proyección paralela ortogonal y su correspondiente encuadre en el cuadro *Modelo de ejemplo*. Al principio, calcula los valores de *ρ (rho)*, *θ (theta)* y *ϕ (phi)*, tal y como se describen en el capítulo 6, apartado 1. Además, para cada vértice del modelo, determina la proyección paralela ortogonal así como el encuadre correspondiente.

Después, calcula las dimensiones del "window" o ventana y las del "viewport" o puerta (ver capítulo 6, apartado 2). Luego, para cada cara del modelo, traza ésta en color gris dibujando el modelo de ejemplo. A continuación, muestra la ventana de *Ejecución* para seleccionar el modo de ejecución (automática o manual), inicializa los cuadros *Resultado del algoritmo Z-Buffer*, *Contenido de los buffers* y *Ecuaciones de los planos visibles* y pasa el control a otra subrutina según el modo de ejecución elegido.

7.2.6. Sub EjecAuto()

Desarrolla la ejecución automática del algoritmo Z-Buffer. Si hay un modelo cargado en memoria y está activa la ejecución automática, pasa el control a *Pseudocodigo*.

7.2.7. Sub InicBuffers()

Inicializa el cuadro *Contenido de los buffers*. Elimina los datos anteriores, rellena las cabeceras del cuadro y limpia las casillas salvo las columnas *X* e *Y* que también las rellena adecuadamente.

7.2.8. Sub InicCodigo()

Inicializa el cuadro *Pseudocódigo Z-Buffer*. Escribe las líneas que forman el pseudocódigo del algoritmo Z-Buffer.

7.2.9. Sub InicEcuPlanos()

Inicializa el cuadro *Ecuaciones de los planos visibles*. Rellena las cabeceras del cuadro.

7.2.10. Sub InicEjecAuto()

Inicializa los menús y variables para la ejecución automática del algoritmo Z-Buffer. Si se pulsa el botón *Automática* o si se hace click en el menú *Ejecución / Automática*, habilita el contador de tiempo y pasa el control a *EjecAuto*.

7.2.11. Sub InicEjecMan()

Inicializa los menús y variables para la ejecución manual del algoritmo Z-Buffer. Si se pulsa el botón *Manual* o si se hace click en el menú *Ejecución / Manual*, deshabilita el contador de tiempo y pasa el foco al cuadro *Pseudocódigo Z-Buffer*.

7.2.12. Sub InicMatFrame()

Inicializa la matriz *Frame[]*. Para cada zbpixel del cuadro *Resultado del algoritmo Z-Buffer*, rellena tanto el frame-buffer como la columna *Frame* del cuadro *Contenido de los buffers* con el color blanco (fondo del cuadro).

7.2.13. *Sub InicMatZBuf()*

Inicializa la matriz *ZBuffer[]*. Para cada zbpixel del cuadro *Resultado del algoritmo Z-Buffer*, rellena tanto el z-buffer como la columna *ZBuffer* del cuadro *Contenido de los buffers*, con el valor "infinito".

7.2.14. *Sub InicResult()*

Inicializa el cuadro *Resultado del algoritmo Z-Buffer*. Traza la cuadrícula de 32x32 casillas en color gris delimitando cada uno de los zbpixels.

7.2.15. *Function iSigPixel(iPixel As Integer) As Integer*

Calcula el pixel siguiente al actual *iPixel*. Basándose en el resultado obtenido por *CalcPixels*, si el pixel *iPixel* está entre el primero y el último de la cara actual, las coordenadas del siguiente son:

$$Y = \text{entero}\left(\frac{\text{iPixel}}{32}\right)$$

$$X = \text{iPixel} - 32 \cdot Y$$

7.2.16. Sub LeeFichero()

Lee el fichero que contiene el modelo de ejemplo. Primero, comprueba si el fichero es válido según aparezca o no la clave *aZb* (ver capítulo 8, apartado 8). En caso de error, realiza el correspondiente tratamiento de errores.

A continuación, mientras no llegue al final del fichero, examina una a una las distintas claves. Si la clave es *V*, lee el número total de vértices que tiene el modelo de ejemplo. Si la clave es *X*, lee las coordenadas 3D de los vértices del modelo.

Si la clave es *C*, lee el número total de caras que tiene el modelo. Si la clave es *L*, lee la lista de los vértices que componen una cara y su color correspondiente.

Si la clave es *O*, lee las coordenadas 3D del observador. Si la clave es *M*, lee las coordenadas 3D del punto de mira.

Si la clave es otra, se produce un error y realiza el correspondiente tratamiento de errores. Finalmente, devuelve el control a *DibModelo*.

7.2.17. Function oVisible(bCara As Byte) As Boolean

Comprueba si la cara *bCara* es o no visible, según el método de Newell, tal y como se describe en el capítulo 6, apartado 3. Para cada vértice de la cara *bCara*, calcula el coeficiente *C* de la ecuación del plano. Si éste es positivo *(C≥0)*, entonces la función devuelve el valor booleano *True*, sino devuelve *False*.

7.2.18. Sub Pseudocodigo()

Recorre el pseudocódigo del algoritmo Z-Buffer en el cuadro correspondiente. Para seguir fácilmente cómo funciona esta subrutina, he aquí el texto del pseudocódigo, tal como aparece en el cuadro:

```
Inicio del ALGORITMO Z-BUFFER
    Inicializar matriz ZBuffer()
    Inicializar matriz Frame()
    Para cada cara del modelo...
        Comprobar visibilidad de la cara
        Para cada pixel de la cara...
            Calcular coordenada Z del pixel
            Si Z < ZBuffer(pixel) entonces...
                ZBuffer(pixel) = Z
                Frame(pixel) = Color del pixel
            Fin Si
        Siguiente pixel
    Siguiente cara
Fin del ALGORITMO Z-BUFFER
```

Por etapas, en *[00] Inicio del ALGORITMO Z-BUFFER*, llama a *CalcDimResult*. Luego, en *[01] Inicializar matriz ZBuffer()* y en *[02] Inicializar matriz Frame()* inicializa las matrices del z-buffer y del frame-buffer, respectivamente.

En *[03] Para cada cara del modelo...* actualiza la línea del pseudocódigo. Si es la última cara, salta hasta la línea *[12]* y finaliza el recorrido. Después, en *[04] Comprobar visibilidad de la cara*, si es visible, determina la ecuación del plano usando *CalcEcuPlano* y calcula los pixels de la cara con *CalcPixels*. Si no es visible, simplemente salta hasta la línea *[12]* y termina el recorrido.

Ya en *[05] Para cada pixel de la cara...* actualiza la línea del pseudocódigo. Si es el último pixel, salta hasta la línea *[11]*. En *[06] Calcular coordenada Z del pixel* utiliza la función *sZPixel* para obtener la componente *Z* del pixel actual.

A continuación, en *[07] Si Z < ZBuffer(pixel) entonces...* compara el valor *Z* recién calculado con el contenido de la matriz *ZBuffer()*. En caso de no verificarse la comparación, salta a la línea *[10]*. Las líneas *[08] ZBuffer(pixel) = Z* y *[09] Frame(pixel) = Color del pixel* son asignaciones que actualizan los contenidos de las matrices. En *[10] Fin Si* termina el bloque *Si...Entonces* que empieza en la línea *[07]*.

Las líneas *[11] Siguiente pixel* y *[12] Siguiente cara* realizan, como su nombre indica, la actualización de los mismos. Termina cuando alcanza la línea *[13] Fin del ALGORITMO Z-BUFFER*.

En caso de que el foco no coincida con la línea actual, se devuelve a la posición correcta (ver capítulo 8, apartado 3).

7.2.19. Function sZPixel(bCara As Byte, bX As Byte, bY As Byte) As Single

Calcula la componente *Z* del zbpixel *(bX,bY)* de la cara *bCara*. Para determinar dicho valor, aplica la ecuación del capítulo 6, apartado 5.

7.2.20. Sub Form_Load()

Carga el formulario base e inicializa las variables y los controles correspondientes. Prepara uno a uno los marcos y los cuadros de cada control. Es la primera subrutina que se ejecuta del programa.

7.2.21. Sub grdBuffers_GotFocus()

Si el cuadro *Contenido de los buffers* tiene el foco, lo pasa al cuadro *Pseudocódigo Z-Buffer*.

7.2.22. Sub grdEcuPlanos_GotFocus()

Si el cuadro *Ecuaciones de los planos visibles* tiene el foco, lo pasa al cuadro *Pseudocódigo Z-Buffer*.

7.2.23. Sub lstCodigo_KeyPress(KeyAscii As Integer)

Si se pulsa una tecla en el cuadro *Pseudocódigo Z-Buffer* y hay un modelo en memoria, verifica si la tecla es *[Intro]* o *[Espacio]*.

Si está activa la ejecución manual y la tecla es *[Intro]*, llama a *Pseudocodigo*. Si está activa la ejecución automática y la tecla es *[Espacio]*, cambia el contador de tiempo y llama a *Pseudocodigo*.

7.2.24. Sub mnuAyudAutor_Click()

Si se hace click en el menú *Ayuda / Autor* o se pulsa la tecla *[F8]*, muestra la ventana de *Autor*.

7.2.25. Sub mnuAyudCont_Click()

Si se hace click en el menú *Ayuda / Contenido* o se pulsa la tecla *[F1]*, muestra la ventana de *Ayuda*.

7.2.26. Sub mnuCambDir_Click()

Si se hace click en el menú *Cambios / Dirección de mira...* o se pulsa la tecla *[F7]*, muestra la ventana de *Cambios*. Si hay cambio de componentes, dibuja el modelo nuevo.

7.2.27. Sub mnuCambMir_Click()

Si se hace click en el menú *Cambios / Punto de mira...* o se pulsa la tecla *[F6]*, muestra la ventana de *Cambios*. Si hay cambio de coordenadas, dibuja el modelo nuevo.

7.2.28. Sub mnuCambObs_Click()

Si se hace click en el menú *Cambios / Observador...* o se pulsa la tecla *[F5]*, muestra la ventana de *Cambios*. Si hay cambio de coordenadas, dibuja el modelo nuevo.

7.2.29. Sub mnuEjecAuto_Click()

Si se hace click en el menú *Ejecución / Automática* o se pulsa la tecla *[F2]*, llama a *InicEjecAuto*.

7.2.30. Sub mnuEjecMan_Click()

Si se hace click en el menú *Ejecución / Manual* o se pulsa la tecla *[F3]*, llama a *DibModelo*.

7.2.31. Sub mnuFichAbrir_Click()

Si se hace click en el menú *Fichero / Abrir...* o se pulsa la tecla *[F4]*, muestra la ventana de *Ficheros* para que el usuario pueda elegir el fichero del modelo. A continuación, llama a *LeeFichero*.

7.2.32. Sub mnuFichSalir_Click()

Si se hace click en el menú *Fichero / Salir* o se pulsa la tecla *[F9]*, cierra el fichero y finaliza la aplicación.

7.2.33. Sub picResult_GotFocus()

Si el cuadro *Resultado del algoritmo Z-Buffer* tiene el foco, lo pasa al cuadro *Pseudocódigo Z-Buffer*.

7.2.34. Sub picResult_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)

Si se mueve el cursor del ratón por el cuadro *Resultado del algoritmo Z-Buffer*, muestra las coordenadas de la situación actual de dicho cursor, asimismo como las del zbpixel que esté señalando.

7.2.35. Sub tmrPaso_Timer()

Activa el contador de tiempo para la ejecución automática del algoritmo Z-Buffer.

7.3. FORM FRMAUTOR

7.3.1. Sub Form_Load()

Carga el formulario base e inicializa las variables y los controles correspondientes. Prepara la imagen de la ventana de *Autor*.

7.3.2. Sub imgAutor_Click()

Si se hace click sobre la imagen que muestra el título del programa y el nombre del autor, descarga la ventana de *Autor*.

7.4. FORM FRMAYUDA

7.4.1. Sub Form_Load()

Carga el formulario base e inicializa las variables y los controles correspondientes. Prepara el cuadro tipo *webbrowser* de la ventana de *Ayuda*, que permite leer un fichero HTML con el contenido de la ayuda del programa (para más información sobre el tema *[Peñ98]*).

7.5. FORM FRMCAMB

7.5.1. Sub cmdNo_Click()

Si se hace click en el botón *No*, descarga la ventana de *Cambios* y deshace cualquier cambio realizado en esta ventana.

7.5.2. Sub cmdSi_Click()

Si se hace click en el botón *Si* y se han realizado cambios en las coordenadas/componentes, éstas se actualizan activándose un indicador de cambio.

7.5.3. Sub Form_Load()

Carga el formulario base e inicializa las variables y los controles correspondientes. Prepara los 3 espacios y diales de las coordenadas y los 2 botones.

7.6. FORM FRMEJEC

7.6.1. Sub cmdAuto_Click()

Si se hace click en el botón *Automática*, activa el *flag* adecuado.

7.6.2. Sub cmdMan_Click()

Si se hace click en el botón *Manual*, activa el *flag* adecuado.

7.6.3. Sub Form_Load()

Carga el formulario base e inicializa las variables y los controles correspondientes. Prepara la ventana de *Ejecución* con los botones *Automática* y *Manual*.

8
MANUAL DEL USUARIO

8. MANUAL DEL USUARIO

El objetivo de este capítulo es mostrar cómo se instala, ejecuta y funciona el programa, cuáles son sus cuadros, menús y ventanas, cuál es la estructura del formato *.aZb*, etc. Se trata, en definitiva, del fichero de ayuda *aZb.htm*, escrito en formato HTML, al que se puede acceder desde la propia aplicación. Sin embargo, se han suprimido los hipervínculos que relacionan los distintos apartados entre sí por su escasa utilidad en un medio como es el papel.

8.1. Instalación y ejecución

En cuanto a la instalación de la aplicación, al estar basado en Windows 95 y habiendo usado el lenguaje Microsoft Visual Basic 5.0, es este entorno de programación el que facilita la tarea creando un proceso de instalación totalmente automatizado. Así, basta con introducir el primero de los dos diskettes en la unidad correspondiente y, desde *Inicio / Panel de control / Agregar o quitar programas*, seguir los pasos que indique el propio sistema. De modo manual, habría que ejecutar el fichero *Setup.exe* del primer diskette desde el *Explorador* de Windows 95. (Por supuesto, como cualquier otro programa de Windows 95 existe un proceso de desinstalación de la aplicación).

Referente a la ejecución del programa, ésta queda reducida a localizar la entrada *aZb* en *Inicio / Programas* de Windows 95 y hacer click con el ratón. Una vez dentro del programa, es necesario abrir un fichero *.aZb* con un modelo para que se pueda ver cómo funciona.

8.2. FUNCIONAMIENTO

8.2.1. Lectura del fichero del modelo

Antes de comenzar a trabajar con el programa, es necesario cargar en la memoria del ordenador un modelo de ejemplo.

Para ello, mediante el menú *Fichero / Abrir...* o pulsando la tecla `[F4]`, se accede a la ventana de ficheros que permite al usuario elegir un fichero con extensión *.aZb* que contiene un modelo.

Al leer el fichero, el programa toma los datos correspondientes al modelo (coordenadas de sus vértices, distribución de sus aristas y colores de sus caras), al observador y al punto de mira (coordenadas de sus posiciones). Además, se comprueba que no exista ningún error de tipo sintáctico (los errores de tipo geométrico *no* pueden verificarse).

8.2.2. Representación del modelo de ejemplo

Cuando termina la lectura del fichero, se visualiza el modelo de alambre en el cuadro *Modelo de ejemplo* y, por medio de la ventana de ejecución, se pregunta al usuario por el modo de ejecución que va a llevarse a cabo.

Esta representación se realiza según una proyección ortogonal paralela y con el encuadre necesario para que se visualice sin hacer uso de *clipping*. Asimismo, se puede observar cómo varía el color de sus aristas conforme se va analizando la visibilidad de las caras del modelo de ejemplo.

8.2.3. Simulación del algoritmo Z-Buffer

Según el modo de ejecución elegido, el usuario puede seguir la simulación del algoritmo por el cuadro *Pseudocódigo Z-Buffer*.

En caso de que haya elegido la ejecución automática, se convertirá en un mero espectador del programa pudiendo observar cómo la barra de color se mueve de una línea a otra siguiendo las instrucciones del algoritmo.

Si eligió la ejecución manual, el usuario es quien decide cuándo pasa de una línea a otra del pseudocódigo pulsando la tecla `[Intro]`.

8.2.4. Cálculo de las ecuaciones de los planos visibles

Una vez que se ha encontrado con una cara visible, el programa añade una línea a la tabla del cuadro *Ecuaciones de los planos visibles* y calcula la ecuación del plano correspondiente a esa cara.

Esta información es útil para que el usuario pueda comprobar que, efectivamente, el coeficiente C es siempre positivo y, además, sepa de qué color es la cara actual.

8.2.5. Cálculo del contenido de los buffers

A continuación, se empiezan a rellenar las distintas filas de la tabla del cuadro *Contenido de los buffers* con los datos calculados correspondientes a la coordenada Z.

Según el valor de esta coordenada, se rellena también el *z-buffer* y se colorea el *frame-buffer*.

8.2.6. Representación del resultado del algoritmo Z-Buffer

Finalmente, el usuario va visualizando la imagen resultante en el cuadro *Resultado del algoritmo Z-Buffer*.

Esta imagen, realizada a muy baja resolución, da una idea bastante aproximada de cómo es el modelo dibujado únicamente con sus caras visibles en sus respectivos colores.

8.3. CUADROS

8.3.1. Resultado del algoritmo Z-Buffer

Posición: Esquina superior izquierda de la ventana principal.

Objetivo: Muestra el resultado de aplicar del algoritmo Z-Buffer al modelo de ejemplo.

Aspecto: Consta de una cuadrícula de 32x32 casillas (*zbpixels*), cada una de las cuales se corresponde con una fila del cuadro *Contenido de los buffers*, que va visualizando los *zbpixels* coloreados de acuerdo con el algoritmo Z-Buffer.

Utilización: El usuario no puede interactuar en este cuadro; sin embargo, puede conocer las coordenadas de un *zbpixel* cualquiera manteniendo el cursor del ratón sobre él.

8.3.2. Contenido de los buffers

Posición: Parte superior central de la ventana principal.

Objetivo: Muestra el contenido del *z-buffer* (o matriz de profundidades) y del *frame-buffer* (o matriz de colores).

Aspecto: Consta de una tabla de 5 columnas y 1025 filas. Las columnas son las coordenadas X, las coordenadas Y, las coordenadas Z, el contenido del z-buffer y el contenido del frame-buffer. Las filas son la cabecera y una por cada *zbpixel* del cuadro *Resultado del algoritmo Z-Buffer*.

Utilización: El usuario puede moverse a lo largo de la tabla mediante la barra de desplazamiento vertical que hay a su derecha. La información correspondiente al *zbpixel* actual aparece en la fila siguiente a la cabecera.

8.3.3. Pseudocódigo Z-Buffer

Posición: Esquina superior derecha de la ventana principal.

Objetivo: Muestra el pseudocódigo del algoritmo Z-Buffer que se va a utilizar en el programa.

Aspecto: Consta de un conjunto de líneas que describe el pseudocódigo y una barra de color que indica la situación en que se encuentra el desarrollo del algoritmo.

Utilización: El usuario puede interactuar en este cuadro dependiendo del modo de ejecución. Si está activa la ejecución automática, pulsando la tecla `[Espacio]` se puede detener/reanudar el recorrido por el pseudocódigo. Si está activa la ejecución manual, pulsando la tecla `[Intro]` se avanza la barra de color de línea en línea a voluntad del usuario.

Nota: Si se observa una mala respuesta por parte del teclado (*pérdida del foco*), basta con hacer *click* con el ratón en cualquier otro cuadro para que el foco vuelva a éste.

8.3.4. Ecuaciones de los planos visibles

Posición: Esquina inferior izquierda de la ventana principal.

Objetivo: Muestra las ecuaciones de los planos/caras visibles del modelo de ejemplo. Una ecuación de plano tiene la forma:

$$A \cdot X + B \cdot Y + C \cdot Z + D = 0$$

Aspecto: Consta de una tabla de 6 columnas y más de 1 fila. Las columnas son el nombre de la cara, el coeficiente A de la ecuación del plano, el coeficiente B, el coeficiente C, el coeficiente D y el color de la cara. Las filas son la cabecera y una por cada plano visible que aparezca en el modelo de ejemplo.

Utilización: El usuario, según se va analizando la visibilidad de cada cara, puede ver cómo se añaden en la tabla fila tras fila bajo la cabecera. Si no se ven todas en la pantalla, se puede mover a lo largo de la tabla mediante la barra de desplazamiento vertical que estará a su derecha.

Nota: Al principio, únicamente se muestra la cabecera de la tabla ya que no hay ninguna cara comprobada todavía.

8.3.5. Modelo de ejemplo

Posición: Esquina inferior derecha de la ventana principal.

Objetivo: Muestra el modelo de ejemplo, en modo de alambre, visto desde la perspectiva indicada con su correspondiente encuadre.

Aspecto: Consta de un cuadrado de 320x320 puntos de resolución que permite saber cuál es la cara en la que está trabajando el algoritmo Z-Buffer.

Utilización: El usuario puede ver marcada en rojo la cara actual del modelo de ejemplo.

Nota: Para evitar posibles vistas complejas de un modelo, antes de representarlo en este cuadro, se calcula su encuadre para que siempre aparezca completo, sin necesidad de realizar *clipping*.

8.4. MENÚS

8.4.1. Fichero

Abrir... Muestra la ventana de ficheros que permite al usuario elegir el fichero que contiene el modelo de ejemplo. Tecla `[F4]`.

Salir Permite la salida del programa. Tecla `[F9]`.

8.4.2. Ejecución

Automática Activa la ejecución automática. Tecla `[F2]`.

Manual Activa la ejecución manual. Tecla `[F3]`.

8.4.3. Cambios

Observador... Muestra la ventana de cambios que permite al usuario cambiar las coordenadas del observador en la escena que está en memoria. Tecla `[F5]`.

Punto de mira... Muestra la ventana de cambios que permite al usuario cambiar las coordenadas del punto de mira en la escena que está en memoria. Tecla `[F6]`.

Dirección de mira... Muestra la ventana de cambios que permite al usuario cambiar las componentes de la dirección de mira en la escena que está en memoria. Tecla `[F7]`.

8.4.4. Ayuda

Contenido Muestra la ventana de ayuda. Tecla `[F1]`.

Autor Muestra la ventana de autor. Tecla `[F8]`.

8.5. VENTANAS

Ventana Principal Muestra los menús y los cuadros de trabajo. Ocupa toda la pantalla.

Ventana de Autor Muestra el título completo del programa y el nombre del autor.

Ventana de Ayuda Muestra la ayuda del programa cuyo contenido se encuentra en el fichero *aZb.htm*.

Ventana de Cambios Permite que el usuario pueda cambiar las coordenadas del observador o del punto de mira o las componentes de la dirección de mira sin necesidad de modificar el fichero *.aZb* del modelo. Se accede a ella mediante el menú *Cambios*.

Ventana de Ejecución Permite que el usuario pueda seleccionar el modo de ejecución al terminar de leer un fichero con formato *.aZb*.

Ventana de Ficheros Permite que el usuario pueda elegir un fichero con formato *.aZb* en una ventana de diálogo típica de Windows. Se accede a ella mediante el menú *Fichero / Abrir...* o pulsando la tecla `[F4]`.

8.6. TECLADO

8.6.1. Teclas de función

[F1] Ayuda / Contenido

[F2] Ejecución / Automática

[F3] Ejecución / Manual

[F4] Fichero / Abrir...

[F5] Cambios / Observador...

[F6] Cambios / Punto de mira...

[F7] Cambios / Dirección de mira...

[F8] Ayuda / Autor

[F9] Fichero / Salir

8.6.2. Otras teclas

[Espacio] Detener/reanudar el recorrido por el cuadro *Pseudocódigo Z-Buffer* cuando está activa la ejecución automática.

[Intro] Avanzar a otra línea del cuadro *Pseudocódigo Z-Buffer* cuando está activa la ejecución manual.

8.7. EJECUCIÓN

8.7.1. Automática

Mediante la ejecución automática, el usuario no interviene en la simulación del algoritmo Z-Buffer, salvo para detener/reanudar el desarrollo de la misma.

Pulsando la tecla `[Espacio]`, se detiene el recorrido por el cuadro *Pseudocódigo Z-Buffer*. Pulsándola nuevamente, continúa la ejecución.

Para activar la ejecución automática, se puede usar el menú *Ejecución / Automática* o pulsar la tecla `[F2]`.

8.7.2. Manual

Mediante la ejecución manual, el usuario puede controlar, paso a paso, la simulación del algoritmo Z-Buffer.

Pulsando la tecla `[Intro]`, se avanza de una línea a otra del cuadro *Pseudocódigo Z-Buffer*.

Para activar la ejecución manual, se puede usar el menú *Ejecución / Manual* o pulsar la tecla `[F3]`.

8.8. Estructura del formato .aZb

8.8.1. Clave aZb

Sintaxis: **AZB**

Definición: Cabecera del fichero en formato **.aZb**.

Posición en el fichero: Primera línea.

8.8.2. Clave V

Sintaxis: **V** número_vértices

Definición: Número de vértices del modelo.

Posición en el fichero: Línea siguiente a la clave **aZb**.

8.8.3. Clave(s) X

Sintaxis: **X** coordenada_X, coordenada_Y, coordenada_Z

Definición: Coordenadas de cada vértice del modelo.

Posición en el fichero: Línea(s) siguiente(s) a la clave **V**.

8.8.4. Clave C

Sintaxis: **C** número_caras

Definición: Número de caras del modelo.

Posición en el fichero: Línea siguiente a la última clave **X**.

8.8.5. Clave(s) L

Sintaxis: **L** número_vértices, vértice_1, ..., vértice_N, color

Definición: Lista(s) de vértices y color de la cara.

Posición en el fichero: Línea(s) siguiente(s) a la clave **C**.

8.8.6. Clave O

Sintaxis: **O** coordenada_X, coordenada_Y, coordenada_Z

Definición: Coordenadas del observador.

Posición en el fichero: Línea siguiente a la última clave **L**.

8.8.7. Clave M

Sintaxis: **M** coordenada_X, coordenada_Y, coordenada_Z

Definición: Coordenadas del punto de mira.

Posición en el fichero: Línea siguiente a la clave **O**.

8.9. LIMITACIONES

- Los ficheros de modelos sólo pueden contener un único modelo.
- Los colores de las caras de los modelos han de ser alguno de los 16 colores básicos de Windows (del 0-Negro al 15-Blanco).
- Los modelos no soportan ningún tipo de texturas.
- El modelo de alambre del cuadro *Modelo de ejemplo* se muestra en perspectiva paralela ortogonal.

9
EJEMPLOS

9. EJEMPLOS

El objetivo de este capítulo es proporcionar un conjunto de ejemplos, suficientemente representativos, para utilizar con el programa. Aquí se describen y visualizan de distintas formas cada uno de ellos.

9.1. TIPOS DE FICHEROS

El usuario puede disponer, inicialmente, de 4 modelos como los que se aparecen a continuación, en 4 condiciones similares cada uno; en total, hay 16 ficheros de ejemplos.

Modelo?.aZb: se trata del fichero original, con el observador en las coordenadas (120, 120, 120) y el punto de mira en las coordenadas (40, 40, 40), es decir, una proyección isométrica.

Modelo?b.aZb: como *Modelo?.aZb*, con la declaración de las caras en orden inverso (de esa manera, puede observarse cómo afecta dicho orden al modelo resultante).

Modelo?c.aZb: como *Modelo?.aZb*, con el observador situado en las coordenadas (50, 120, 100).

Modelo?x.aZb: como *Modelo?.aZb*, con algún error en el fichero (falta o sobra un vértice, falta una cara, la clave aZb está incompleta,...).

9.2. MODELO 1

9.2.1. Descripción

El *Modelo 1* es un poliedro cóncavo con 20 vértices, 12 caras y 30 aristas. Consta de 6 heptágonos y 6 triángulos.

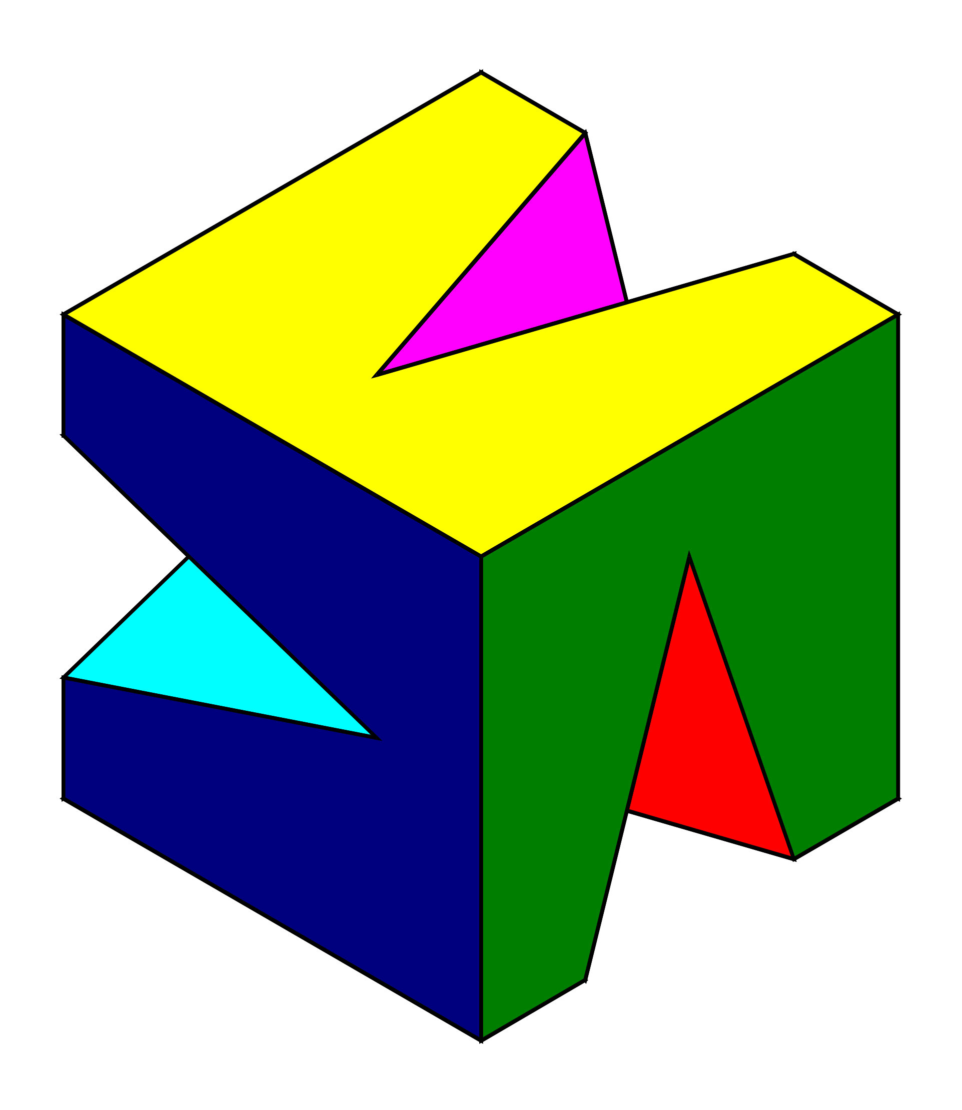
La descripción de las coordenadas de sus vértices así como de las caras y de los vértices que las componen se encuentra en las siguientes tablas:

Modelo 1 - Vértices										
	V00	***V01***	***V02***	***V03***	***V04***	***V05***	***V06***	***V07***	***V08***	***V09***
X	0	80	80	20	80	80	0	0	0	0
Y	0	0	0	0	0	0	0	20	40	60
Z	0	0	20	40	60	80	80	80	20	80
	V10	***V11***	***V12***	***V13***	***V14***	***V15***	***V16***	***V17***	***V18***	***V19***
X	0	0	80	80	60	40	20	80	60	40
Y	80	80	80	80	80	80	80	60	40	20
Z	80	0	80	0	0	60	0	40	80	0

Modelo 1 - Planos												
	P00	***P01***	***P02***	***P03***	***P04***	***P05***	***P06***	***P07***	***P08***	***P09***	***P10***	***P11***
1º	V00	V00	V00	V01	V02	V03	V05	V07	V08	V10	V14	V15
2º	V01	V06	V11	V13	V17	V17	V12	V18	V18	V12	V19	V19
3º	V02	V07	V16	V12	V03	V04	V10	V08	V09	V13	V15	V16
4º	V03	V08	V19	V05			V09			V14		
5º	V04	V09	V14	V04			V18			V15		
6º	V05	V10	V13	V17			V07			V16		
7º	V06	V11	V01	V02			V06			V11		
Col	10	09	06	01	11	04	14	13	03	02	05	12

(Esta página está intencionadamente en blanco)

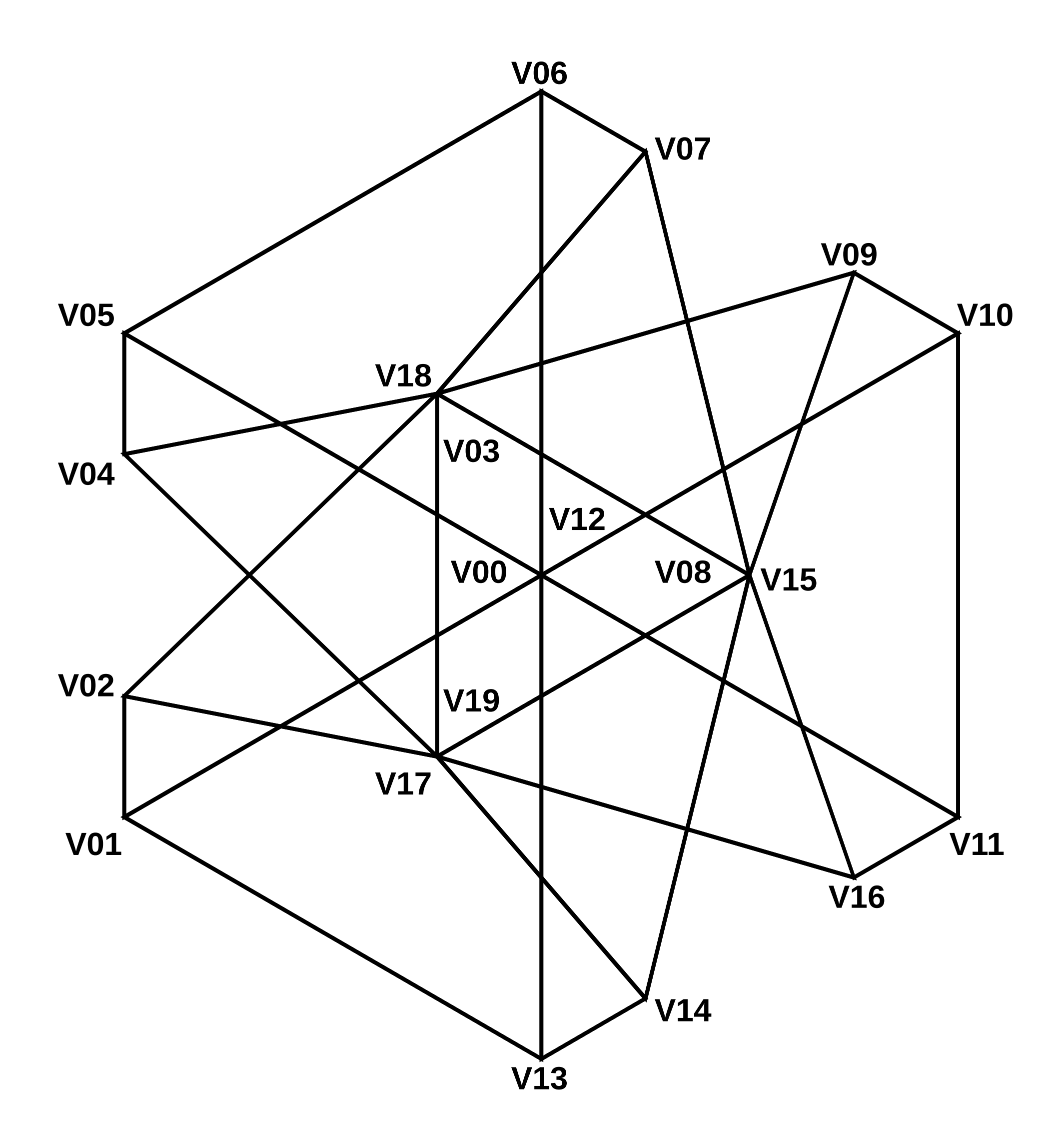
9.2.2. Modelo original





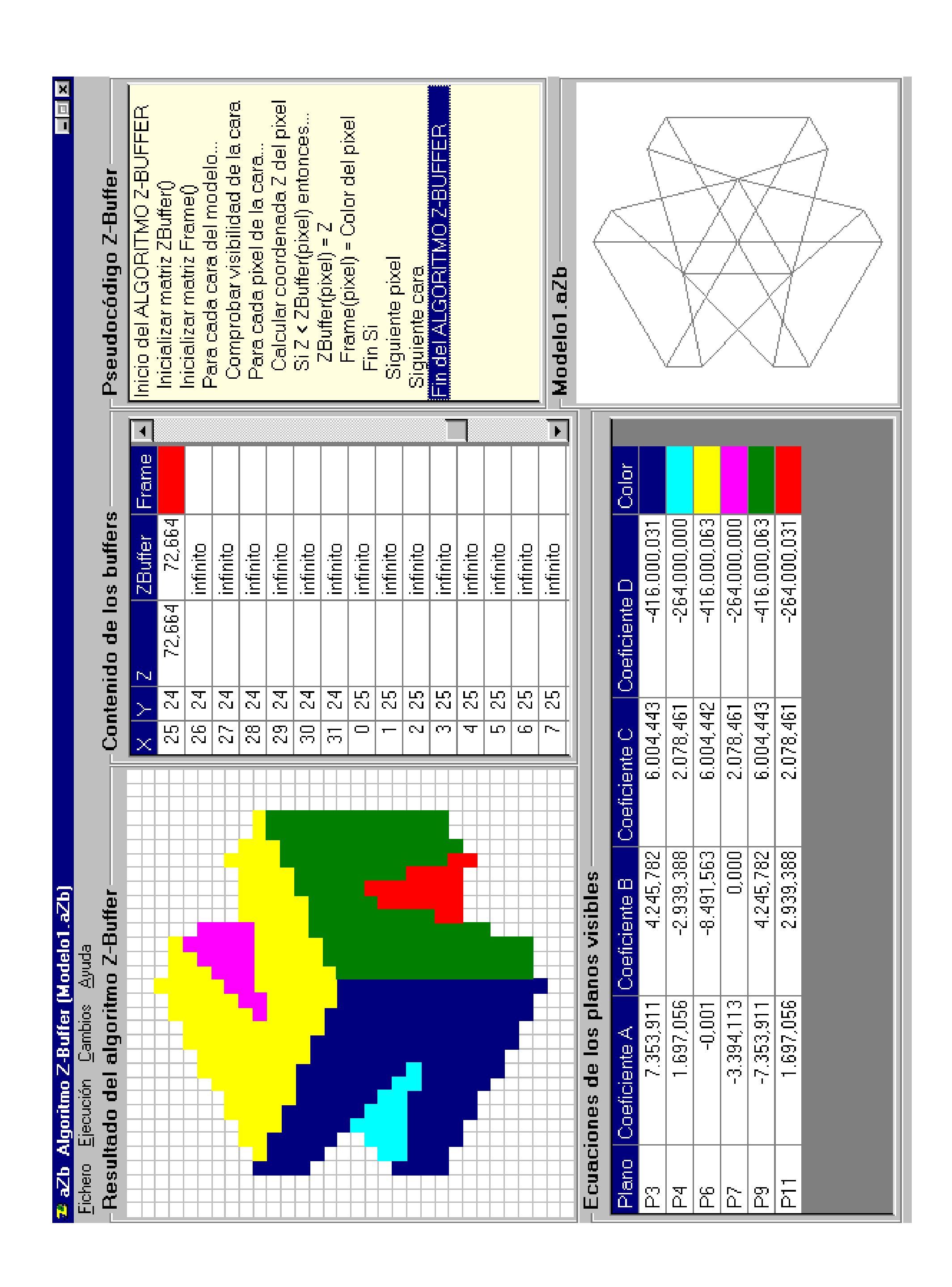
(Esta página está intencionadamente en blanco)

9.2.3. Modelo de alambre



(Esta página está intencionadamente en blanco)

9.2.4. Pantalla del programa



(Esta página está intencionadamente en blanco)

9.3. MODELO 2

9.3.1. Descripción

El *Modelo 2* es un poliedro cóncavo con 17 vértices, 11 caras y 26 aristas. Consta de 2 heptágonos, 1 hexágono y 8 cuadriláteros.

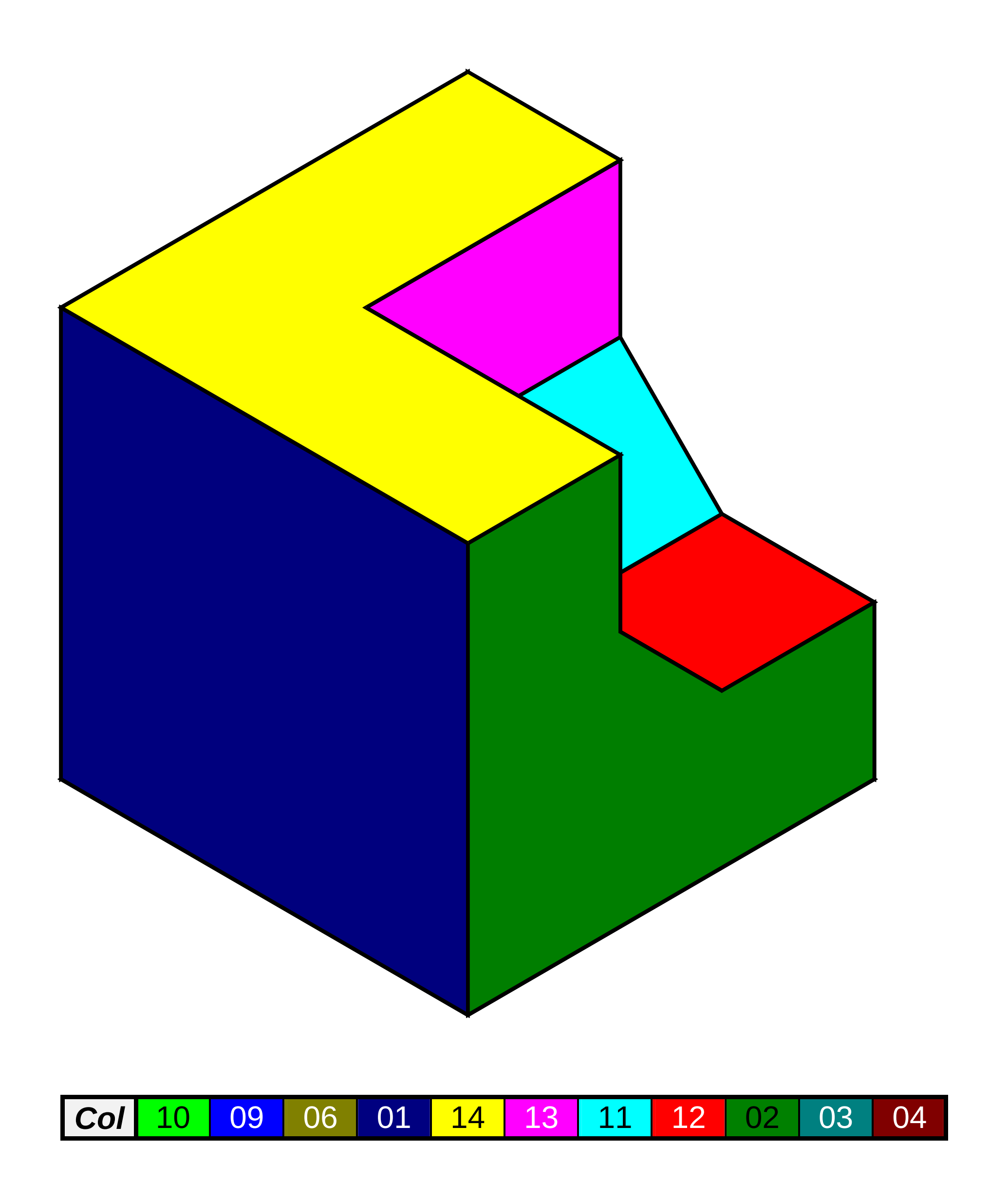
La descripción de las coordenadas de sus vértices así como de las caras y de los vértices que las componen se encuentra en las siguientes tablas:

Modelo 2 - Vértices									
	V00	***V01***	***V02***	***V03***	***V04***	***V05***	***V06***	***V07***	***V08***
X	0	80	80	0	0	0	0	0	0
Y	0	0	0	0	30	30	50	80	80
Z	0	0	80	80	80	50	30	30	0
	V09	***V10***	***V11***	***V12***	***V13***	***V14***	***V15***	***V16***	
X	30	50	50	80	80	50	50	30	
Y	80	80	80	80	80	30	30	50	
Z	30	50	50	80	0	80	50	30	

Modelo 2 - Planos											
	P00	***P01***	***P02***	***P03***	***P04***	***P05***	***P06***	***P07***	***P08***	***P09***	***P10***
1º	V00	V00	V00	V01	V02	V04	V05	V06	V07	V09	V10
2º	V01	V03	V08	V13	V12	V14	V15	V16	V09	V16	V15
3º	V02	V04	V13	V12	V11	V15	V16	V09	V10	V15	V14
4º	V03	V05	V01	V02	V14	V05	V06	V07	V11	V10	V11
5º		V06			V04				V12		
6º		V07			V03				V13		
7º		V08							V08		
Col	10	09	06	01	14	13	11	12	02	03	04

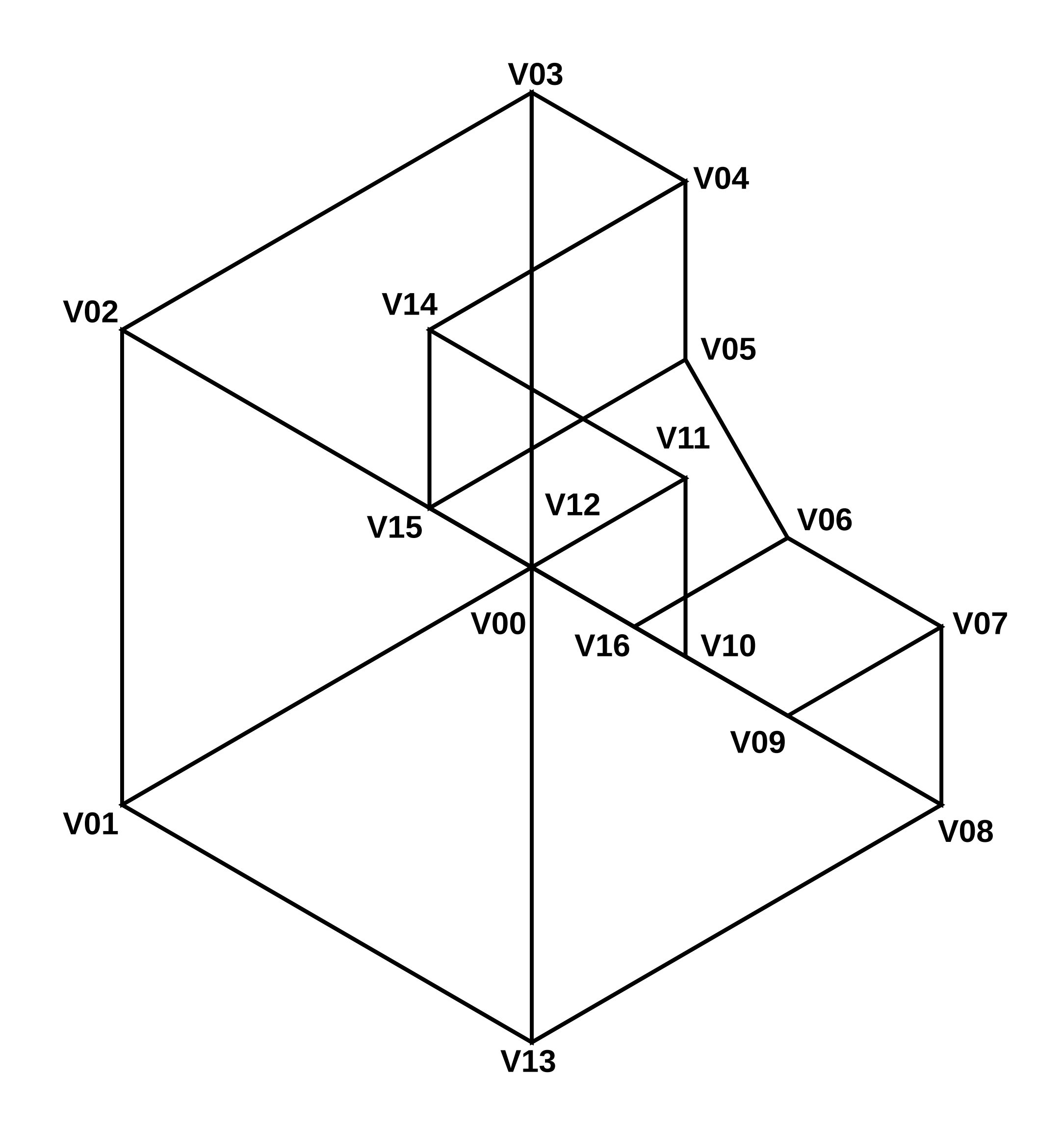
(Esta página está intencionadamente en blanco)

9.3.2. Modelo original



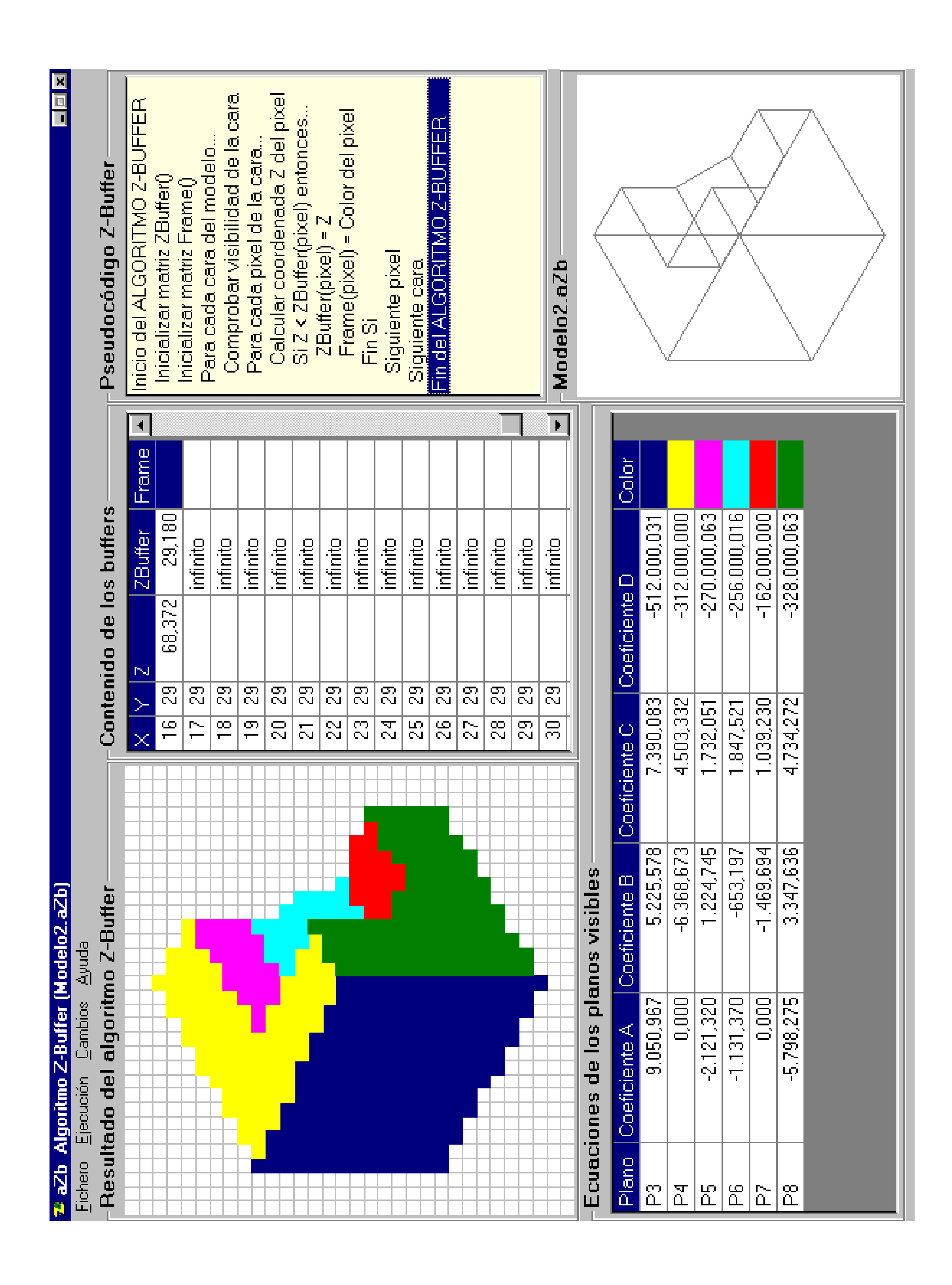
(Esta página está intencionadamente en blanco)

9.3.3. Modelo de alambre



(Esta página está intencionadamente en blanco)

9.3.4. Pantalla del programa



(Esta página está intencionadamente en blanco)

9.4. MODELO 3

9.4.1. Descripción

El *Modelo 3* es un poliedro cóncavo con 10 vértices, 10 caras y 18 aristas. Consta de 2 pentágonos, 2 cuadriláteros y 6 triángulos.

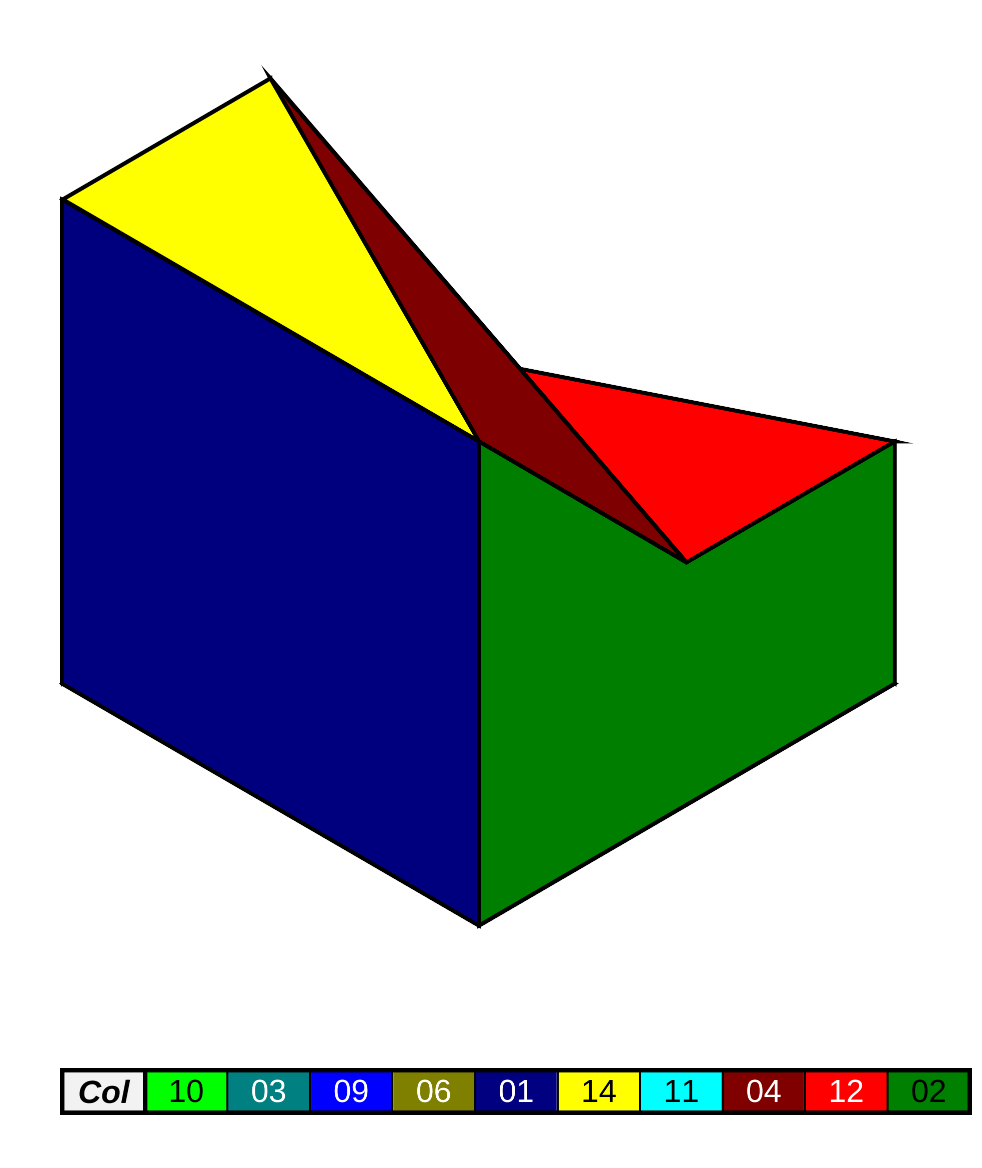
La descripción de las coordenadas de sus vértices así como de las caras y de los vértices que las componen se encuentra en las siguientes tablas:

Modelo 3 - Vértices										
	V00	**V01**	**V02**	**V03**	**V04**	**V05**	**V06**	**V07**	**V08**	**V09**
X	0	80	80	40	40	0	0	40	80	80
Y	0	0	0	0	0	80	80	80	80	80
Z	0	0	80	80	40	40	0	40	80	0

Modelo 3 - Planos										
	P00	**P01**	**P02**	**P03**	**P04**	**P05**	**P06**	**P07**	**P08**	**P09**
1º	V00	V00	V00	V00	V01	V02	V03	V03	V04	V05
2º	V01	V04	V05	V06	V09	V08	V07	V08	V07	V07
3º	V02	V05	V06	V09	V08	V03	V04	V07	V05	V08
4º	V03			V01	V02					V09
5º	V04									V06
Col	10	03	09	06	01	14	11	04	12	02

(Esta página está intencionadamente en blanco)

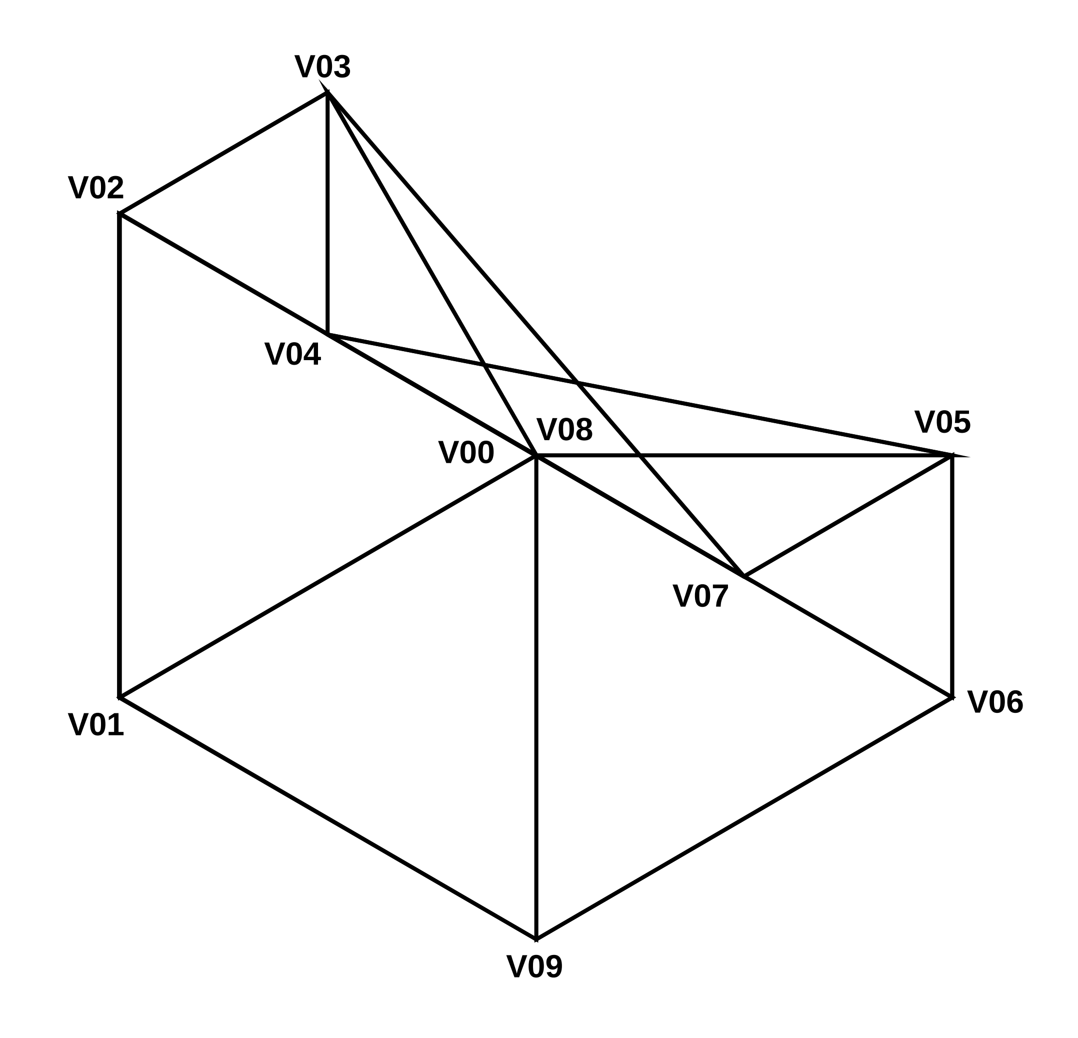
9.4.2. Modelo original



Nota: como podrá verse en la imagen del apartado 9.4.4, el resultado obtenido no coindice *plenamente* con este modelo original. El motivo se debe a las aproximaciones y redondeos al sustituir en la ecuación del plano y a la baja resolución del cuadro *Resultado* ....

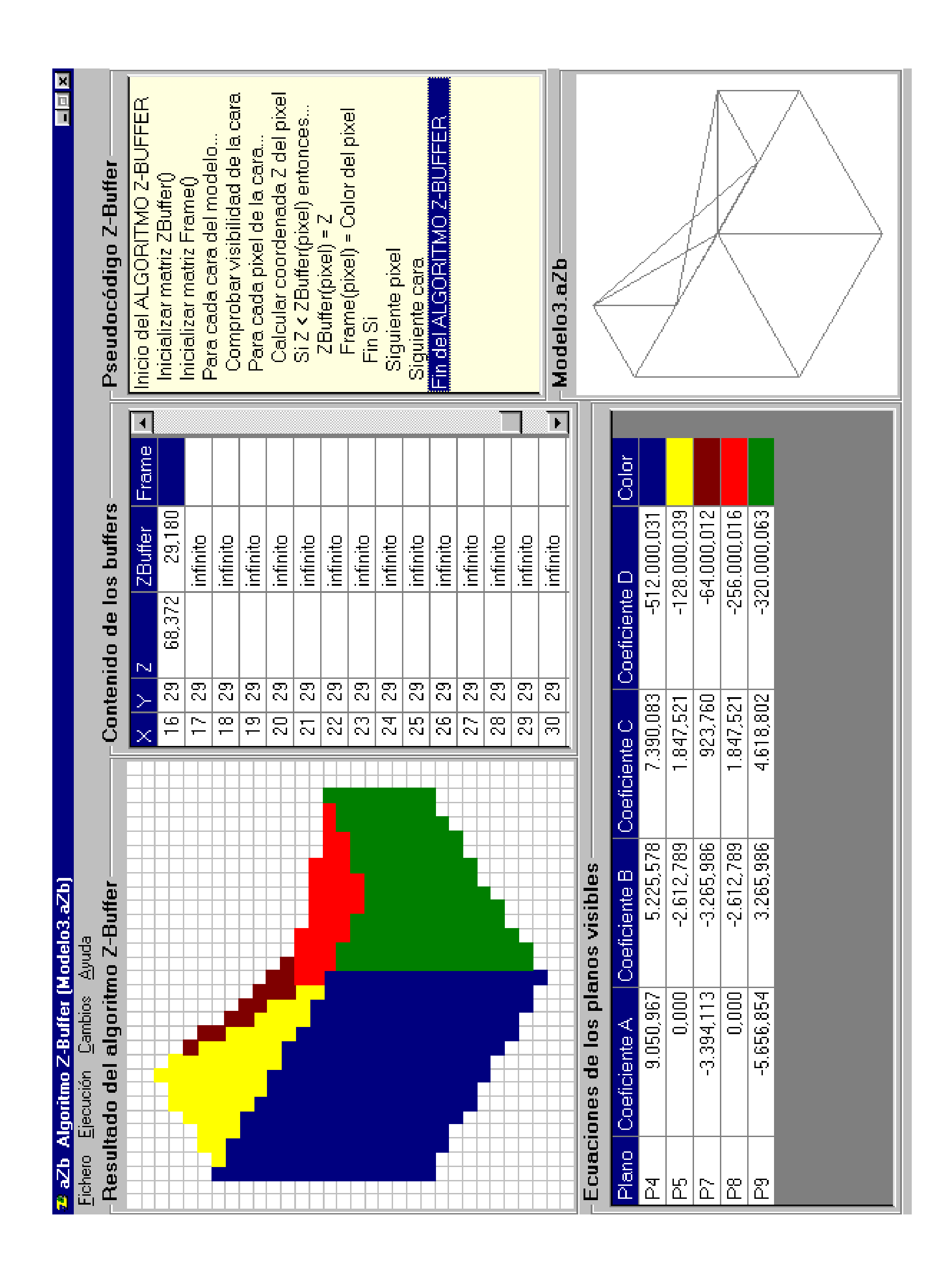
(Esta página está intencionadamente en blanco)

9.4.3. Modelo de alambre



(Esta página está intencionadamente en blanco)

9.4.4. Pantalla del programa



(Esta página está intencionadamente en blanco)

9.5. MODELO 4

9.5.1. Descripción

El *Modelo 4* es un poliedro convexo con 13 vértices, 10 caras y 21 aristas. Consta de 3 hexágonos, 3 cuadriláteros y 4 triángulos.

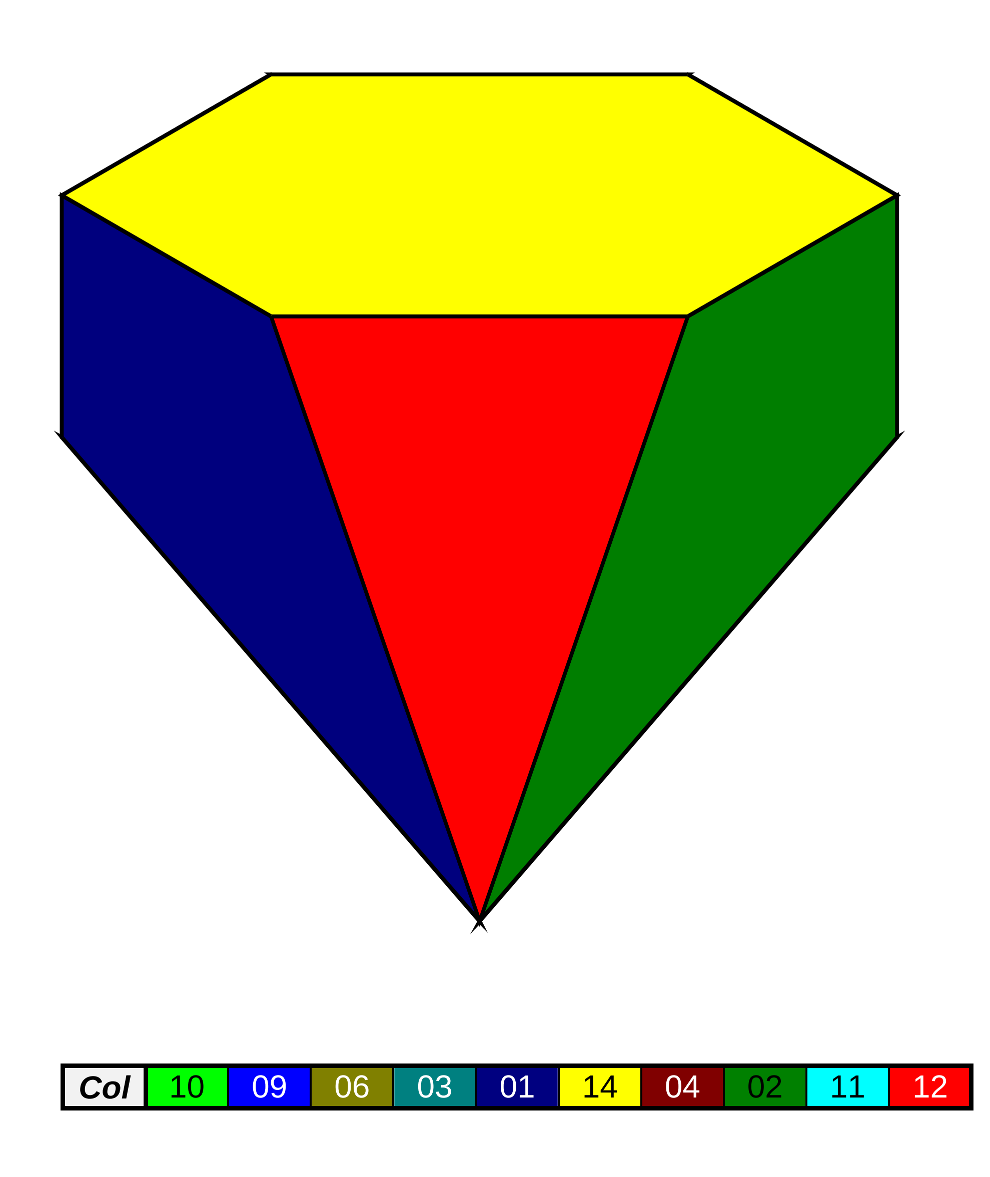
La descripción de las coordenadas de sus vértices así como de las caras y de los vértices que las componen se encuentra en las siguientes tablas:

Modelo 4 - Vértices													
	V00	***V01***	***V02***	***V03***	***V04***	***V05***	***V06***	***V07***	***V08***	***V09***	***V10***	***V11***	***V12***
X	0	40	80	80	40	0	0	0	0	0	40	80	80
Y	0	0	0	0	0	0	40	80	80	40	80	80	40
Z	0	0	40	80	80	40	80	80	40	0	80	0	80

Modelo 4 - Planos										
	P00	***P01***	***P02***	***P03***	***P04***	***P05***	***P06***	***P07***	***P08***	***P09***
1º	V00	V00	V00	V01	V02	V03	V04	V07	V08	V10
2º	V01	V05	V09	V11	V11	V12	V06	V10	V11	V12
3º	V02	V06	V11	V02	V12	V10	V05	V11	V09	V11
4º	V03	V07	V01		V03	V07		V08		
5º	V04	V08				V06				
6º	V05	V09				V04				
Col	10	09	06	03	01	14	04	02	11	12

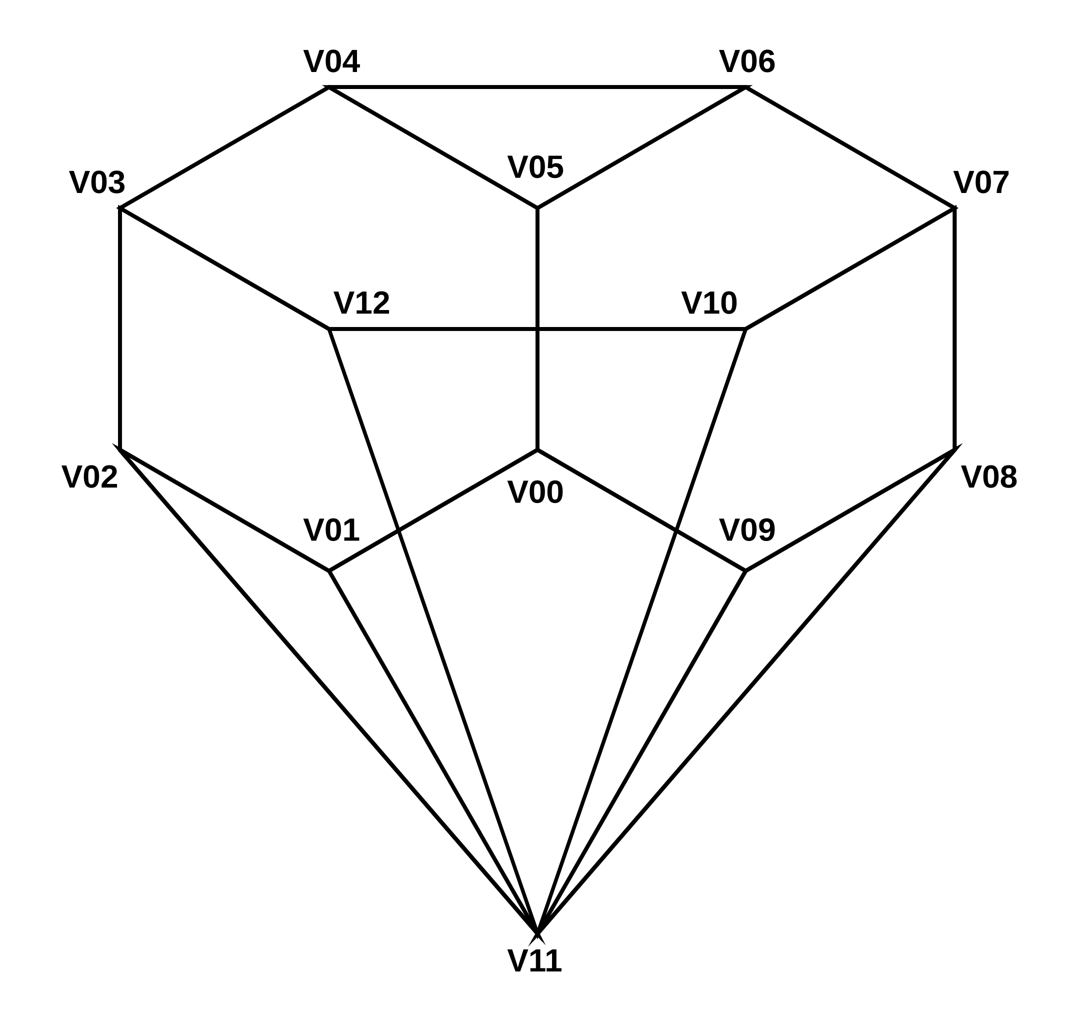
(Esta página está intencionadamente en blanco)

9.5.2. Modelo original



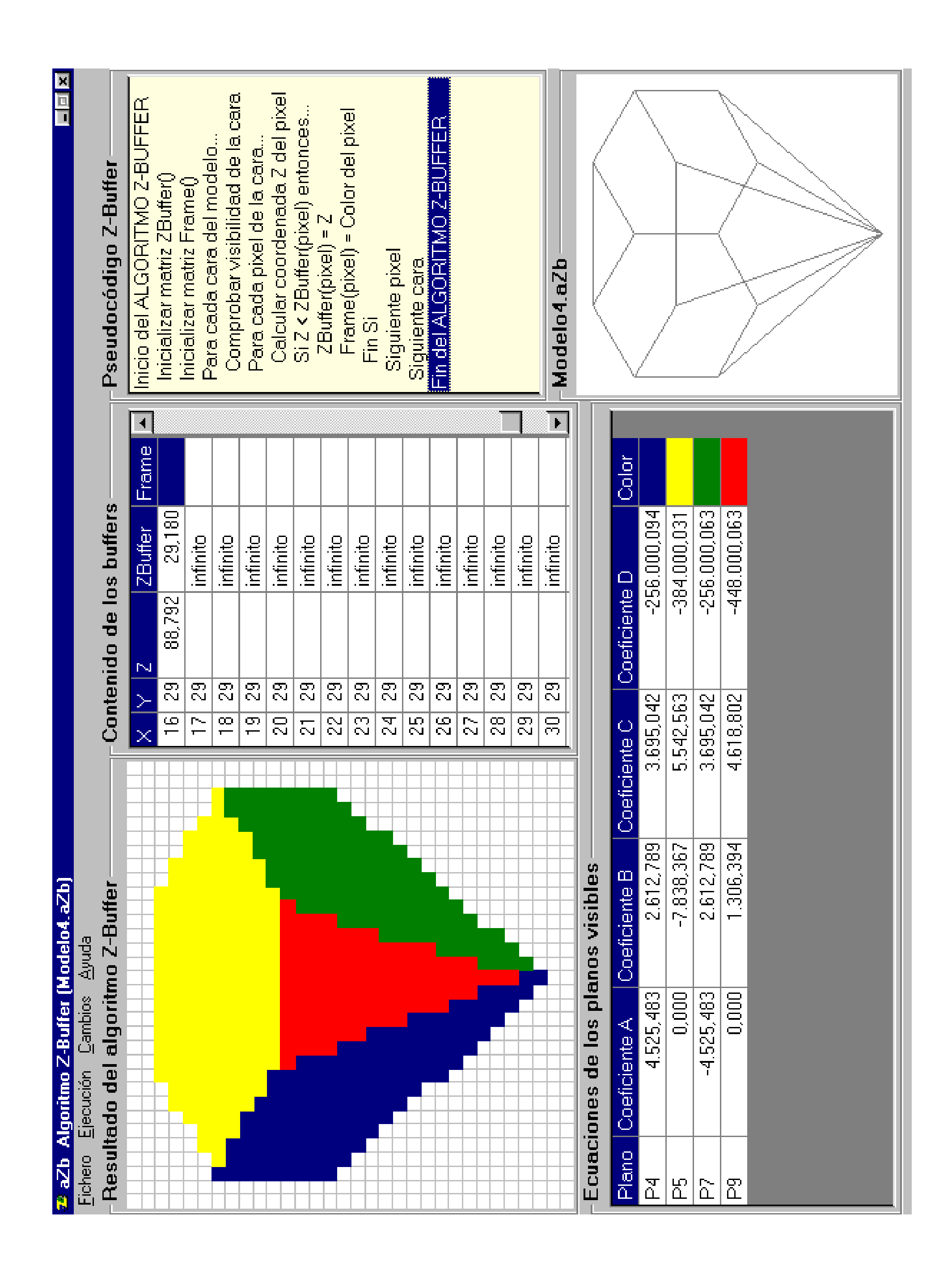
(Esta página está intencionadamente en blanco)

9.5.3. Modelo de alambre



(Esta página está intencionadamente en blanco)

9.5.4. Pantalla del programa



10
CONCLUSIONES Y DESARROLLOS FUTUROS

10. CONCLUSIONES Y DESARROLLOS FUTUROS

El objetivo de este capítulo es contemplar las conclusiones finales del proyecto y los posibles desarrollos futuros, es decir, las mejoras del programa y la ampliación de éste para el uso de otros algoritmos.

10.1. CONCLUSIONES

En primer lugar, su aspecto didáctico/pedagógico al que estaba enfocado desde su concepción ha sido tratado, tanto en el aspecto visual como en el interactivo usuario/máquina, cumpliéndose con lo estipulado en las especificaciones del capítulo 3.

Pese a la necesidad de establecer ciertos cambios irrelevantes en el propio pseudocódigo del algoritmo Z-Buffer para facilitar la comprensión y la utilización interactiva por parte del usuario y pese a que los modelos de ejemplos son simples, puede decirse que el programa deja ver sus múltiples aplicaciones en el campo educativo ya que la explicación de ciertos cálculos y procedimientos llevados a cabo por el algoritmo se pueden observar aquí con total nitidez, sin necesidad de necesitar mucho tiempo para obtener resultados concretos.

Además, se observa que esta aplicación permite exponer sin ninguna dificultad conceptos, como el uso de buffers o matrices por parte del algoritmo que, en una pizarra, no es fácil de explicar. Asimismo, la posibilidad de examinar el contenido de cada una de las matrices de manera interactiva permite que el usuario sepa en todo momento qué está ocurriendo.

Otra conclusión que queda bastante clara también es el uso del ordenador en el campo educativo. Sin el desarrollo de aplicaciones como ésta, comprender mediante la lectura de un libro cómo funciona un algoritmo de eliminación de superficies ocultas es bastante árido. Ahora bien, si en lugar de limitarnos a leer cómo es el algoritmo *vemos* cómo *funciona* éste de manera interactiva, el grado de aprendizaje crecerá de manera exponencial. (Suele decirse que "una imagen vale más que mil palabras").

10.2. DESARROLLOS FUTUROS

En cuanto a los posibles desarrollos futuros existen 2 enfoques bastante claros. El primero sería las distintas mejoras al programa en sí y el segundo, la ampliación al uso de otros algoritmos.

10.2.1. Mejoras del programa

- Ficheros multimodelos. Aunque en otra parte de esta obra se han expuesto los motivos por los cuales se utiliza un único modelo por fichero, también es lógico suponer que, en ciertas ocasiones, sea necesario poder aplicar el algoritmo a escenas más complejas para ver cómo se comportan las partes de un modelo que solapa a otro.

- Uso de texturas y color verdadero (*true color*). Al igual que en el caso anterior, también es útil disponer de la posibilidad de usar texturas para cada una de las caras de los modelos así como poder emplear millones de colores (actualmente, la aplicación soporta únicamente 16 colores, suficientes para su uso didáctico).

Para admitir las mejoras anteriores, sería necesario revisar el formato de los ficheros *.aZb* y designar más claves o, en su defecto, modificar las que ya existen. Por ejemplo, la clave *L* incorpora al final el color de la cara con un número que va del 0 (color negro) al 15 (color blanco, color del fondo). Bastarían pocos cambios para permitir el uso de un mayor intervalo de colores (millones quizás).

- Modelos no poliédricos. Una limitación del programa es la exclusiva utilización de figuras poliédricas, no permitiéndose el empleo de modelos con superficies de revolución o superficies alabeadas. Sin embargo, al estar enfocado a la resolución del problema de eliminación de superficies ocultas usando el algoritmo Z-Buffer, este inconveniente no es tal debido a que dicho algoritmo se comporta excelentemente en superficies planas de modelos poliédricos, no así en otro tipo de superficies.

- Alta resolución. Otra mejora es la posibilidad de que el usuario disponga de varios tamaños de cuadrículas a la hora de representar el resultado. Así, en lugar de las 32x32 casillas actuales del cuadro *Resultado del algoritmo Z-Buffer*, en algunos casos, se podrían emplear cuadrículas menores (por ejemplo, de 20x20 casillas) o mayores (de 48x48 casillas). Con esto, se tiene la ventaja de que

para ciertos modelos y vistas los posibles errores, consecuencia de la baja resolución, se subsanarían adecuadamente; mientras que en los casos en que se usen modelos demasiado simples, al reducir el número de casillas se reduce también el tiempo de ejecución obteniéndose un resultado igualmente satisfactorio.

- Mayor información en el cuadro *Resultado del algoritmo Z-Buffer*. En esta primera versión, se han utilizado unos indicadores que permitían al usuario saber las coordenadas de la casilla en que se hallaba el cursor del ratón simplemente pasando éste por encima de dicha casilla. Sin embargo, esta posibilidad podría ampliarse mostrando más información como puede ser el valor de la componente Z y el color del frame-buffer del pixel que se está señalando. De esta manera, el cuadro *Contenido de los buffers* sería superfluo, permitiendo así disponer de más superficie en la pantalla del ordenador para mostrar imágenes más grandes.

- Cambios interactivos de coordenadas. Esta mejora se refiere a la imposibilidad actual de ver cómo quedaría una vista al modificar sus coordenadas/componentes en la ventana de *Cambios*. Como habrá podido observar el usuario, al acceder a dicha ventana, sólo está permitido variar los valores de las coordenadas pero hasta que no se verifican pulsando el botón *Si*, no se puede ver el nuevo resultado. Con esta mejora, se podría tener una idea de cómo va a quedar antes de admitirla, sin posibilidad de sorpresas posteriores como es que el modelo esté muy sesgado y las caras sean poco visibles.

Hay que destacar que muchas de estas mejoras serían para un uso profesional del programa en lugar del uso educativo dado aquí. El motivo es que ciertas mejoras serían absurdas puesto que harían perder su simplicidad al programa a la hora de exponer los distintos conceptos en que se basa el funcionamiento del algoritmo. Como ejemplo, si la imagen resultante fuese en alta resolución (600x600 pixels), esta sería más acorde con la realidad; sin embargo, el coste en tiempo de ejecución así como en memoria sería disparatado.

10.2.2. Ampliación al uso de otros algoritmos

- Elección de algoritmos dentro de la misma aplicación. Una idea sería añadir un nuevo menú que permitiera al usuario elegir, dentro de un abanico de posibilidades, qué algoritmo va a utilizar para resolver el problema de la eliminación de superficies ocultas. Por supuesto, al tener que utilizar un mismo interface para cualquier algoritmo disponible, la capacidad didáctica no quedaría muy ajustada, sin embargo, podrían compararse entre sí los distintos algoritmos.

- Distintas aplicaciones similares a la actual. En sentido totalmente opuesto al anterior y, quizás la mejor solución, sería realizar varias aplicaciones similares en cuanto a su filosofía de exposición y pedagogía, pero adaptando cada una al algoritmo que se va a tratar. De esta manera, los interfaces serían los más adecuados. Como complemento, se podría realizar un entorno que permitiera el uso en ventanas independientes de las diversas aplicaciones.

- Determinar cuáles son los algoritmos idóneos. Esta no es una cuestión baladí, ya que hay algoritmos que no son fáciles de implementar (el algoritmo que los describe no puede explicarse en pocas palabras) mientras que otros tienen la particularidad de que son difíciles de exponer de manera interactiva y pedagógica.

11
BIBLIOGRAFÍA

11. BIBLIOGRAFÍA

El objetivo de este capítulo es recopilar en una lista la documentación empleada, tanto títulos de libros como artículos de revistas y páginas web en Internet. No es una bibliografía exhaustiva pero sí puede servir de complemento al lector.

11.1. LIBROS

[Ada89] Adams, Lee

High-Performance CAD Graphics in C.

Windcrest, 1989.

[Bur89] Burger, Peter / Gillies, Duncan

Interactive Computer Graphics.

Addison Wesley, 1989.

[Cor85] Cortés Parejo, José

Introducción al Tratamiento de Gráficos por Ordenador.

Publicaciones de la Universidad de Sevilla, 1985.

[Cor93] Cortés Parejo, José

Gráficos por Ordenador: Técnicas y Métodos.

Universidad de Sevilla, 1993.

[Don86] Dony, Robert

Grafismo Científico con Microordenador.

Masson, 1986.

[Don88] Dony, Robert

Eliminación de Partes Ocultas.

Masson, 1988.

[Fer86] Ferrer Abelló, Antonio M.* (editor)*.

Dibujar con el Ordenador.

Biblioteca Básica Informática Nº 15. Ingelek, 1986.

[Fol90] Foley, J.D. / van Dam, A. / Feiner, S.K. / Hughes, J.F.

Computer Graphics: Principles and Practice.

Addison Wesley, 1990.

[Her96] Hernández Rodríguez, Fco. / Martín Navarro, Antº.

Técnicas de Representación Gráfica mediante Computador II.

Dpto. de Ingeniería del Diseño. Universidad de Sevilla, 1996.

[Mar97] Martín, Luis / Martín, Juan Manuel

Cómo Programar en Visual Basic.

Biblioteca Técnica de Programación Nº 4. Prensa Técnica, 1997.

[New79] Newman, William M. / Sproull, Robert F.

Principles of Interactive Computer Graphics.

McGraw-Hill, 1979.

[Sel88] Sellarés i Chiva, Joan Antoni

Fundamentos de los Gráficos con Ordenador.

EDUNSA, 1988.

[Wat89] Watt, Alan

Fundamentals of Three-Dimensional Computer Graphics.

Addison Wesley, 1989.

11.2. REVISTAS

[Ala95] Alarcón, Enrique de

Programación de Entornos Virtuales - Cómo Calcular Puntos 3D.

Tower Communications. Sólo Programadores 13, 1995. Pp. 48-52.

[Ang97] Anguiano, Eloy

Perspectiva - Z-Buffer.

IDG Communications. PC World España 129, 1997. Pp. 100-101.

[Ant96] Antón Alonso, Pedro

Programación de Entornos Virtuales - Ocultación de Caras en 3D.

Tower Communications. Sólo Programadores 22, 1996. Pp. 64-68.

[Fer94] Fernández, Luis Fernando

Programación de Entornos Virtuales - Movimiento Fluido en 3D.

Tower Communications. Sólo Programadores 3, 1994. Pp. 14-20.

[Fer95] Fernández, Luis Fernando

Programación de Entornos Virtuales - Ocultación de Objetos.

Tower Communications. Sólo Programadores 6, 1995. Pp. 16-21.

[Mon95] Montero, Ramón

Tutor - Conceptos Básicos de Gráficos en 3D.

IDG Communications. PC World España 116, 1995. Pp. 284-298.

[Mon96a] Montero, Ramón

Tutor - Definición de Objetos Gráficos 3D: Modelado.

IDG Communications. PC World España 118, 1996. Pp. 198-210.

[Mon96b] Montero, Ramón

Tutor - Gráficos 3D: Imágenes Finales.

IDG Communications. PC World España 121, 1996. Pp. 278-289.

[Peñ98] Peña Tresancos, Jaime

Programación - ¿Cómo incluir un navegador en sus programas?

IDG Communications. PC World España 140, 1998. Pp. 249-257.

[Rui98] Ruiz, Antonio

Entornos 3D - El Algoritmo Z-Buffer.

Prensa Técnica. Programación Actual 10, 1998. Pp. 30-33.

11.3. DOCUMENTOS EN INTERNET

[Loi95] Loisel, Sebastien

A Tutorial for 2D & 3D Vector and Texture Mapped Graphics 0.50ß.

http://www.math.mcgill.ca/~loisel/

[Mor94] Mortensen, Zach (Voltaire/OTM)

OTMPOLY.DOC: Complete HOW TO of Polygons.

http://www.geocities.com/SiliconValley/Park/9784/otmpoly.txt

ANEXO A
CÓDIGO FUENTE

A. CÓDIGO FUENTE

El objetivo de este anexo es exponer la totalidad del código fuente, en Visual Basic, utilizado por la aplicación. Esto es, el contenido de los ficheros *aZb.bas*, *aZb.frm*, *aZbAutor.frm*, *aZbAyuda.frm*, *aZbCamb.frm* y *aZbEjec.frm*.

No obstante, por motivos de formateado al transcribirlo, se han modificado levemente algunas líneas sin que por ello afecte en absoluto al entendimiento y/o funcionamiento del programa.

A.1. FICHERO AZB.BAS (MÓDULO MDLAZB)

```
' aZb  Algoritmo Z-Buffer
' -----------------------
' Módulo: mdlAZB (Principal).
' Autor: Antonio José Torres Rodríguez de Almansa.
' ==================================================================

Option Explicit



'--- Constantes globales -----------------------------------------

Public Const KbNP As Byte = 0                      'nombre del plano
Public Const KbA As Byte = 1     'coef. A de la ecuación del plano
Public Const KbB As Byte = 2     'coef. B de la ecuación del plano
Public Const KbC As Byte = 3     'coef. C de la ecuación del plano
Public Const KbD As Byte = 4     'coef. D de la ecuación del plano
Public Const KbCP As Byte = 5                       'color del plano

Public Const KbX As Byte = 0       'componente X de una coordenada
Public Const KbY As Byte = 1       'componente Y de una coordenada
Public Const KbZ As Byte = 2       'componente Z de una coordenada
Public Const KbZB As Byte = 3              'contenido del Z-Buffer
Public Const KbFB As Byte = 4          'contenido del frame-buffer

Public Const KiErrTipo As Integer = vbOKOnly + _
 vbExclamation                          'tipo de cuadro de error

Public Const KlFlags As Long = cdlOFNHideReadOnly + _
 cdlOFNFileMustExist + cdlOFNExplorer + _
 cdlOFNNoLongNames           '"flags" para cuadro Abrir fichero...

Public Const KsVMinR As Single = 1.55              'valor mínimo del
                                        '"viewport" del resultado
Public Const KsVMaxR As Single = 31 - KsVMinR     'valor máximo del
                                        '"viewport" del resultado
Public Const KsVMinM As Single = 15.95             'valor mínimo del
                                           '"viewport" del modelo
Public Const KsVMaxM As Single = 319 - KsVMinM   'valor máximo del
                                       'del "viewport" del modelo

Public Const KtAZBTit As String = _
 "aZb  Algoritmo Z-Buffer"                       'título principal
Public Const KtModTit As String = _
 "Modelo de ejemplo"               'título del cuadro del modelo
```

```
'--- Estructuras globales ----------------------------------------

'--- estructura del tipo Modelo

Public Type TModelo
  bNTV As Byte               'número de vértices totales del modelo
  sV3D() As Single       'coordenadas 3D de los vértices originales
  sV3P() As Single      'coordenadas 3D de los vértices proyectados
  sV2EM() As Single      'coordenadas 2D de los vértices del modelo
  bV2ER() As Byte     'coordenadas 2D de los vértices del resultado
  bNTC As Byte                            'caras totales del modelo
  bLVC() As Byte                    'lista de vértices de cada cara
  bCol() As Byte                                'color de cada cara
  sEcP() As Single       'ecuaciones del plano de cada cara visible
End Type

Public GTmMod As TModelo

'--- estructura del tipo Cámara

Public Type TCamara
  sObs(KbX To KbZ) As Single           'coordenadas del observador
  sMir(KbX To KbZ) As Single        'coordenadas del punto de mira
  sDir(KbX To KbZ) As Single  'componentes de la dirección de mira
End Type

Public GTcCam As TCamara

'--- estructura del tipo Buffers

Public Type TBuffers
  bFrame(0 To 31, 0 To 31) As Byte       'matriz frame-buffer o de
                                                          'colores
  sZBuf(0 To 31, 0 To 31) As Single         'matriz z-buffer o de
                                                    'profundidades
  oCara(0 To 31, 0 To 31) As Boolean     'matriz de pixels de una
                                                             'cara
End Type

Public GTbBuf As TBuffers
```

```
'--- Variables globales -----------------------------------------

Public GbCA As Byte                                'cara actual del modelo
Public GbFont As Byte                       'tamaño de la fuente de letras
Public GbLA As Byte                         'línea actual del pseudocódigo
Public GbPX As Byte                         'coordenada X del pixel actual
Public GbPY As Byte                         'coordenada Y del pixel actual

Public GiPA As Integer                           'pixel actual de la cara
Public GiPP As Integer                          'pixel primero de la cara
Public GiPU As Integer                           'pixel último de la cara
Public GiTPC As Integer                     '1 por ciento de la pantalla

Public GoCambio As Boolean               'hay o no cambio de coordenadas
Public GoModelo As Boolean                'hay o no un modelo en memoria

Public GsDim As Single                     'dimensión máxima de encuadre
Public GsDX As Single, GsDY As Single             'diferencia en X e Y
Public GsMinX As Single, GsMaxX As Single            'valores mínimo y
                                                          'máximo de X
Public GsMinY As Single, GsMaxY As Single            'valores mínimo y
                                                          'máximo de Y
Public GsMaxZ As Single                            'valor máximo de Z
Public GsPI As Single       'definición del número PI = 3.14159265
Public GsWMinX As Single, GsWMaxX As Single          'valores mínimo y
                                          'máximo de X en el "window"
Public GsWMinY As Single, GsWMaxY As Single          'valores mínimo y
                                          'máximo de Y en el "window"
Public GsZPA As Single                      'componente Z del pixel actual

Public GtBoton As String                     'botón de ejecución pulsado
Public GtCamb As String                                  'tipo de cambio
Public GtEjec As String                               'modo de ejecución
Public GtErrText As String                  'texto de la ventana de error
Public GtErrTit As String                  'título de la ventana de error
Public GtFichTit As String                 'nombre de fichero en memoria
Public GtMenu As String                        'opción del menú Ejecución
```

A.2. FICHERO AZB.FRM (FORM FRMAZB)

```
' aZb  Algoritmo Z-Buffer
' -----------------------
' Formulario: frmAZB (Principal).
' Autor: Antonio José Torres Rodríguez de Almansa.
' ================================================================

Option Explicit



Private Sub CalcDimResult()

' Cálculo de las dimensiones del "viewport" y de las coordenadas
'   del resultado del Z-Buffer
' ----------------------------------------------------------------

  Dim bV As Byte                                 'contador de vértices

  With GTmMod

    '--- para cada vértice del modelo, calcula coordenadas X e Y

    ReDim .bV2ER(0 To .bNTV - 1, KbX To KbY)

    For bV = 0 To .bNTV - 1
      .bV2ER(bV, KbX) = ((KsVMaxR - KsVMinR) * .sV3P(bV, KbX) + _
       (KsVMinR * GsWMaxX - KsVMaxR * GsWMinX)) / _
       (GsWMaxX - GsWMinX)                                'calcula X

      .bV2ER(bV, KbY) = 31 - (((KsVMaxR - KsVMinR) * _
       .sV3P(bV, KbY) + (KsVMinR * GsWMaxY - KsVMaxR * GsWMinY)) _
       / (GsWMaxY - GsWMinY))                             'calcula Y
    Next bV

  End With

End Sub
```

```
Private Sub CalcEcuPlano(bCara As Byte)

' Cálculo de la ecuación del plano de la cara actual, según Newell
'-------------------------------------------------------------------

  Dim bNV As Byte                    'número de vértices de la cara
  Dim bV As Byte                              'contador de vértices
  Dim bVA As Byte, bVS As Byte  'vértices actual y siguiente de la
                                                              'cara
  Dim sA As Single, sB As Single         'coeficientes A y B de la
                                     'ecuación del plano de la cara
  Dim sC As Single, sD As Single         'coeficientes C y D de la
                                     'ecuación del plano de la cara
  Dim tForm As String     'formato de las columnas de coeficientes

  With GTmMod

    '--- para cada vértice de la cara, calcula A y B

    sA = 0: sB = 0: sD = 0
    bNV = .bLVC(bCara, 0)

    For bV = 1 To bNV
      bVA = .bLVC(bCara, bV)
      bVS = IIf(bV <> bNV, .bLVC(bCara, bV + 1), .bLVC(bCara, 1))
      sA = sA + (.sV3P(bVA, KbY) - .sV3P(bVS, KbY)) * _
       (.sV3P(bVA, KbZ) + .sV3P(bVS, KbZ))              'calcula A
      sB = sB + (.sV3P(bVA, KbZ) - (.sV3P(bVS, KbZ)) * _
       (.sV3P(bVA, KbX) + .sV3P(bVS, KbX))              'calcula B
    Next bV

    .sEcP(bCara, KbA) = sA: .sEcP(bCara, KbB) = sB
    sC = .sEcP(bCara, KbC)
    sD = -(sA * .sV3P(bVS, KbX) + sB * .sV3P(bVS, KbY) + _
     sC * .sV3P(bVS, KbZ))                              'calcula D

    '--- rellena el cuadro de ecuaciones de planos visibles

    With grdEcuPlanos
      .Rows = .Rows + 1
      .Row = .Rows - 1
      .TextMatrix(.Row, KbNP) = "P" & CStr(bCara)
      tForm = "###,##0.000"
      .TextMatrix(.Row, KbA) = Format(sA, tForm)
      .TextMatrix(.Row, KbB) = Format(sB, tForm)
      .TextMatrix(.Row, KbC) = Format(sC, tForm)
      .TextMatrix(.Row, KbD) = Format(sD, tForm)
      .Col = KbCP
      .CellBackColor = QBColor(GTmMod.bCol(bCara) Mod 15)
    End With

  End With

End Sub
```

```
Private Sub CalcPixels(bCara As Byte)

' Cálculo de los pixels correspondientes a una cara
' -----------------------------------------------------------------

  Dim bELB As Byte                   'extremo de una línea de barrido
  Dim bLB() As Byte                    'matriz de la línea de barrido
  Dim bNV As Byte                      'número de vértices de la cara
  Dim bV As Byte                                'contador de vértices
  Dim bVA As Byte, bVS As Byte  'vértices actual y siguiente de la
                                                                 'cara
  Dim bXAux As Byte, bYAux As Byte            'variables auxiliares
  Dim bXMin As Byte, bXMax As Byte   'X mínima y máxima de la cara
  Dim bYMin As Byte, bYMax As Byte   'Y mínima y máxima de la cara
  Dim bX As Byte, bY As Byte                  'contadores de bucles
  Dim bJ As Byte, iY As Integer               'contadores de bucles
  Dim iDX As Integer, iDY As Integer         'diferencias de X e Y
  Dim sY As Single                       'línea de barrido auxiliar

  With GTmMod

    '--- inicializa la matriz de pixels de la cara

    For bY = 0 To 31
      For bX = 0 To 31
        GTbBuf.oCara(bX, bY) = False
      Next bX
    Next bY

    '--- para cada vértice de la cara, traza la arista con el
    '      vértice siguiente

    bNV = .bLVC(bCara, 0)

    For bV = 1 To bNV
      bVA = .bLVC(bCara, bV)
      bVS = IIf(bV <> bNV, .bLVC(bCara, bV + 1), .bLVC(bCara, 1))
      bYMin = .bV2ER(bVA, KbY): bYMax = .bV2ER(bVS, KbY)
      If bYMin > bYMax Then _
       bYAux = bYMin: bYMin = bYMax: bYMax = bYAux
      GTbBuf.oCara(.bV2ER(bVA, KbX), .bV2ER(bVA, KbY)) = True

      For bY = bYMin + 1 To bYMax - 1
        iDX = CInt(.bV2ER(bVS, KbX)) - CInt(.bV2ER(bVA, KbX))
        iDY = CInt(.bV2ER(bVS, KbY)) - CInt(.bV2ER(bVA, KbY))
        bX = .bV2ER(bVA, KbX) + _
         (CInt(bY) - CInt(.bV2ER(bVA, KbY))) * (iDX / iDY)
        GTbBuf.oCara(bX, bY) = True
      Next bY

      GTbBuf.oCara(.bV2ER(bVS, KbX), .bV2ER(bVS, KbY)) = True

    Next bV
```

```
'--- para cada vértice de la cara, calcula las Y mínima y
'     máxima

bXMin = 31: bXMax = 0: bYMin = 31: bYMax = 0

For bV = 1 To bNV
  bVA = .bLVC(bCara, bV)
  bXAux = .bV2ER(bVA, KbX): bYAux = .bV2ER(bVA, KbY)
  If (bYAux < bYMin Or (bYAux = bYMin And bXAux < bXMin)) Then
    bXMin = bXAux: bYMin = bYAux
  End If
  If (bYAux > bYMax Or (bYAux = bYMax And bXAux > bXMax)) Then
    bXMax = bXAux: bYMax = bYAux
  End If
Next bV

GiPP = 32 * bYMin + bXMin            'calcula el primer pixel
GiPU = 32 * bYMax + bXMax            'calcula el último pixel

'--- para cada línea de barrido de la cara, rellena la cara

For bY = bYMin To bYMax
  sY = CSng(bY)

  '--- comprueba si la línea de barrido coincide con vértices

  For bV = 1 To bNV
    bVA = .bLVC(bCara, bV): bYAux = .bV2ER(bVA, KbY)
    If bY = bYAux Then _
     sY = IIf(bY <> bYMax, CSng(bY + 0.5), CSng(bY - 0.5))
  Next bV

  '--- para cada vértice de la cara, calcula los elementos de
  '     la línea de barrido

  ReDim bLB(1 To 2)
  bELB = 0

  For bV = 1 To bNV
    bVA = .bLVC(bCara, bV)
    bVS = IIf(bV <> bNV, .bLVC(bCara, bV + 1), _
    .bLVC(bCara, 1))

    If (.bV2ER(bVA, KbY) < sY And sY < .bV2ER(bVS, KbY)) Or _
     (.bV2ER(bVA, KbY) > sY And sY > .bV2ER(bVS, KbY)) Then
      iDX = CInt(.bV2ER(bVS, KbX)) - CInt(.bV2ER(bVA, KbX))
      iDY = CInt(.bV2ER(bVS, KbY)) - CInt(.bV2ER(bVA, KbY))
      bELB = bELB + 1
      If bELB > UBound(bLB, 1) Then _
       ReDim Preserve bLB(1 To bELB)
      bLB(bELB) = .bV2ER(bVA, KbX) + _
       (CInt(bY) - CInt(.bV2ER(bVA, KbY))) * (iDX / iDY)
    End If

  Next bV
```

```
      '--- ordena la matriz de la línea de barrido

      For bX = 2 To bELB
        For iY = bELB To bX Step -1
          If bLB(iY) < bLB(iY - 1) Then
            bYAux = bLB(iY): bLB(iY) = bLB(iY - 1): _
             bLB(iY - 1) = bYAux
          End If
        Next iY
      Next bX

      '--- corrige la posibilidad de duplicidad de pixels en
      '     líneas múltiples

      If bELB < 2 Then
        For iY = 1 To bELB Step 2
          If iY <> 1 Then
            If bLB(iY) = bLB(iY - 1) Then bLB(iY) = bLB(iY) + 1
          End If
        Next iY
      End If

      '--- rellena la matriz de pixels de la cara

      For bJ = 1 To bELB Step 2
        For bX = bLB(bJ) To bLB(bJ + 1)
          GTbBuf.oCara(bX, bY) = True
        Next bX
      Next bJ

    Next bY

  End With

End Sub
```

```
Private Sub DibCara(bCara As Byte, bColor As Byte)

' Dibujo de una cara del modelo
' ------------------------------------------------------------------

  Dim bNV As Byte                   'número de vértices de la cara
  Dim bV As Byte                             'contador de vértices
  Dim bVA As Byte, bVS As Byte  'vértices actual y siguiente de la
                                                             'cara

  With GTmMod

    '--- para cada vértice de la cara, dibuja la arista con el
    '     vértice siguiente

    bNV = .bLVC(bCara, 0)

    For bV = 1 To bNV
      bVA = .bLVC(bCara, bV)
      bVS = IIf(bV <> bNV, .bLVC(bCara, bV + 1), .bLVC(bCara, 1))
      picModelo.DrawStyle = vbSolid
      picModelo.Line (.sV2EM(bVA, KbX), .sV2EM(bVA, KbY))- _
       (.sV2EM(bVS, KbX), .sV2EM(bVS, KbY)), vbWhite
      picModelo.Line (.sV2EM(bVA, KbX), .sV2EM(bVA, KbY))- _
       (.sV2EM(bVS, KbX), .sV2EM(bVS, KbY)), QBColor(bColor)
    Next bV

  End With

End Sub
```

```
Private Sub DibModelo()

' Dibujo del modelo mediante proyección paralela ortogonal y
'   encuadre
' -----------------------------------------------------------------

  Dim bC As Byte                                   'contador de caras
  Dim bI As Byte                                   'contador de bucle
  Dim bV As Byte                                'contador de vértices
  Dim sRho As Single    'distancia del observador al punto de mira
  Dim sTheta As Single, sPhi As Single    'ángulos de posición del
                                                       'observador
  Dim sSenP As Single, sCosP As Single       'seno y coseno de Phi
  Dim sSenT As Single, sCosT As Single     'seno y coseno de Theta
  Dim sSTSP As Single, sSTCP As Single       'variables auxiliares
  Dim sCTSP As Single, sCTCP As Single       'variables auxiliares

  picModelo.Cls                        'limpia el cuadro del modelo

  GtErrTit = "Error: Dibujo del Modelo"

  With GTcCam

    '--- calcula la distancia Rho

    sRho = 0

    If Not (GoCambio And GtCamb = "D") Then
      For bI = KbX To KbZ
        .sDir(bI) = .sObs(bI) - .sMir(bI)
        sRho = sRho + .sDir(bI) * .sDir(bI)
      Next bI
    Else                 'si hay cambios en la dirección de mira...
      For bI = KbX To KbZ
        sRho = sRho + .sDir(bI) * .sDir(bI)
      Next bI
    End If
    sRho = Sqr(sRho)                                    'calcula Rho

    '--- calcula los ángulos Theta y Phi según corresponda

    GsPI = 4 * Atn(1)                              'PI = 3.14159265

    If .sDir(KbX) = 0 Then                         'calcula si XD=0
      If .sDir(KbY) = 0 Then                'calcula si XD=0 y YD=0
        If .sDir(KbZ) = 0 Then      'calcula si XD=0, YD=0 y ZD=0
          GtErrText = "Observador igual que punto de mira"
          If MsgBox(GtErrText, KiErrTipo, GtErrTit) = vbOK Then _
           Exit Sub                                'mensaje de error
        Else                        'calcula si XD=0, YD=0 y ZD<>0
          sTheta = GsPI
          sPhi = Sgn(.sDir(KbZ)) * GsPI / 2
        End If
```

```
      Else                                          'calcula si XD=0 y YD<>0
        sTheta = Sgn(.sDir(KbY)) * GsPI / 2
        sPhi = Atn(.sDir(KbZ) / Sqr(.sDir(KbX) * .sDir(KbX) + _
        .sDir(KbY) * .sDir(KbY)))
      End If
    Else                                                   'calcula si XD<>0
      sTheta = Atn(.sDir(KbY) / .sDir(KbX))
      sPhi = Atn(.sDir(KbZ) / Sqr(.sDir(KbX) * .sDir(KbX) + _
       .sDir(KbY) * .sDir(KbY)))
    End If

  End With

  '--- calcula los valores trigonométricos necesarios

  sSenT = Sin(sTheta): sCosT = Cos(sTheta)
  sSenP = Sin(sPhi): sCosP = Cos(sPhi)
  sSTSP = sSenT * sSenP: sSTCP = sSenT * sCosP
  sCTSP = sCosT * sSenP: sCTCP = sCosT * sCosP

  With GTmMod

    '--- para cada vértice del modelo, calcula la proyección
    '      paralela ortogonal

    ReDim .sV3P(0 To .bNTV - 1, KbX To KbZ)

    GsMaxZ = 0

    For bV = 0 To .bNTV - 1
      .sV3P(bV, KbX) = -sSenT * (.sV3D(bV, KbX) - _
       GTcCam.sMir(KbX)) + sCosT * (.sV3D(bV, KbY) - _
       GTcCam.sMir(KbY))                                   'calcula X
      .sV3P(bV, KbY) = -sCTSP * (.sV3D(bV, KbX) - _
       GTcCam.sMir(KbX)) - sSTSP * (.sV3D(bV, KbY) - _
       GTcCam.sMir(KbY)) + sCosP * (.sV3D(bV, KbZ) - _
       GTcCam.sMir(KbZ))                                   'calcula Y
      .sV3P(bV, KbZ) = -sCTCP * (.sV3D(bV, KbX) - _
       GTcCam.sMir(KbX)) - sSTCP * (.sV3D(bV, KbY) - _
       GTcCam.sMir(KbY)) - sSenP * (.sV3D(bV, KbZ) - _
       GTcCam.sMir(KbZ)) + sRho                            'calcula Z
      If .sV3P(bV, KbZ) > GsMaxZ Then GsMaxZ = .sV3P(bV, KbZ)
    Next bV

    GsMaxZ = GsMaxZ * 10                            'calcula Z máxima

    '--- para cada vértice del modelo, calcula el encuadre

    GsMinX = .sV3P(0, KbX): GsMaxX = GsMinX
    GsMinY = .sV3P(0, KbY): GsMaxY = GsMinY
```

```
    For bV = 1 To .bNTV - 1
      GsMinX = IIf(.sV3P(bV, KbX) <= GsMinX, .sV3P(bV, KbX), _
       GsMinX)                                    'calcula X mínima
      GsMaxX = IIf(.sV3P(bV, KbX) >= GsMaxX, .sV3P(bV, KbX), _
       GsMaxX)                                    'calcula X máxima
      GsMinY = IIf(.sV3P(bV, KbY) <= GsMinY, .sV3P(bV, KbY), _
       GsMinY)                                    'calcula Y mínima
      GsMaxY = IIf(.sV3P(bV, KbY) >= GsMaxY, .sV3P(bV, KbY), _
       GsMaxY)                                    'calcula Y máxima
    Next bV

    '--- calcula las dimensiones del "window"

    GsDX = GsMaxX - GsMinX: GsDY = GsMaxY - GsMinY
    GsDim = IIf(GsDX >= GsDY, GsDX, GsDY)
    GsWMinX = GsMinX - (GsDim - GsDX) / 2
    GsWMaxX = GsMaxX + (GsDim - GsDX) / 2
    GsWMinY = GsMinY - (GsDim - GsDY) / 2
    GsWMaxY = GsMaxY + (GsDim - GsDY) / 2

    '--- para cada vértice del modelo, calcula las dimensiones del
    '     "viewport"

    ReDim .sV2EM(0 To .bNTV - 1, KbX To KbY)

    For bV = 0 To .bNTV - 1
      .sV2EM(bV, KbX) = ((KsVMaxM - KsVMinM) * .sV3P(bV, KbX) + _
       (KsVMinM * GsWMaxX - KsVMaxM * GsWMinX)) / _
       (GsWMaxX - GsWMinX)                               'calcula X
      .sV2EM(bV, KbY) = 319 - (((KsVMaxM - KsVMinM) * _
       .sV3P(bV, KbY) + (KsVMinM * GsWMaxY - _
       KsVMaxM * GsWMinY)) / (GsWMaxY - GsWMinY))        'calcula Y
    Next bV

    '--- para cada cara del modelo, dibuja la cara

    For bC = 0 To .bNTC - 1
      Call DibCara(bC, 7)        'dibuja la cara del modelo (gris)
    Next bC

  End With

  If GtMenu = "" Then _
   frmEjec.Show vbModal          'muestra el mensaje de ejecución

  mnuEjec.Enabled = True               'menú Ejecución disponible
  mnuCamb.Enabled = True                'menú Cambios disponible

  Call InicResult  'inicializa el cuadro de resultado del Z-Buffer
  Call InicBuffers   'inicializa el cuadro de contenido de buffers
  Call InicEcuPlanos    'inicializa el cuadro de las ecuaciones de
                                           'los planos visibles
```

```
  '--- si se pulsó el botón Automática...

  If GtBoton = "A" Then _
   Call InicEjecAuto           'inicializa la ejecución automática

  '--- si se pulsó el botón Manual o se hizo click en el menú
  '     Ejecución/Manual...

  If GtBoton = "M" Or GtMenu = "M" Then _
   Call InicEjecMan                'inicializa la ejecución manual

End Sub



Private Sub EjecAuto()

' Desarrollo de la ejecución automática del algoritmo Z-Buffer
' -----------------------------------------------------------------

  '--- si hay un modelo en memoria y está activa la ejecución
  '     automática...

  If GoModelo And GtEjec = "A" Then
    Call Pseudocodigo        'recorre el pseudocódigo del Z-Buffer
  End If

End Sub
```

```
Private Sub InicBuffers()

' Inicialización del cuadro de contenido de los buffers
' -----------------------------------------------------------------

  Dim bI As Byte, bJ As Byte                'contadores de bucles

  Screen.MousePointer = vbHourglass   'cambia el cursor a un reloj

  With grdBuffers

    '--- rellena las cabeceras del cuadro de contenido de buffers

    .Clear                            'elimina los datos anteriores

    .ColWidth(KbX) = GiTPC * 3: .TextMatrix(0, KbX) = "X"
    .ColWidth(KbY) = .ColWidth(KbX): .TextMatrix(0, KbY) = "Y"
    .ColWidth(KbZ) = GiTPC * 7.5: .TextMatrix(0, KbZ) = "Z"
    .ColWidth(KbZB) = .ColWidth(KbZ): _
     .TextMatrix(0, KbZB) = "ZBuffer"
    .ColWidth(KbFB) = GiTPC * 6: .TextMatrix(0, KbFB) = "Frame"

    '--- para cada casilla de X e Y, inicializa el cuadro

    .Col = KbFB

    For bJ = 0 To 31
      For bI = 0 To 31
        .Row = (32 * bJ + bI) + 1
        .TextMatrix(.Row, KbX) = bI
        .TextMatrix(.Row, KbY) = bJ
        .TextMatrix(.Row, KbZ) = ""
        .TextMatrix(.Row, KbZB) = ""
        .CellBackColor = QBColor(8)
      Next bI
    Next bJ

    .TopRow = 1       'coloca la primera línea en la parte superior

  End With

  Screen.MousePointer = vbDefault    'cambia el cursor al original

End Sub
```

```
Private Sub InicCodigo()

' Inicialización del cuadro de pseudocódigo del algoritmo Z-Buffer
' ------------------------------------------------------------------

  '--- escribe el pseudocódigo del algoritmo Z-Buffer

  With lstCodigo

    .List(0) = "Inicio del ALGORITMO Z-BUFFER"
    .List(1) = "  Inicializar matriz ZBuffer()"
    .List(2) = "  Inicializar matriz Frame()"
    .List(3) = "  Para cada cara del modelo..."
    .List(4) = "    Comprobar visibilidad de la cara"
    .List(5) = "    Para cada pixel de la cara..."
    .List(6) = "      Calcular coordenada Z del pixel"
    .List(7) = "      Si Z < ZBuffer(pixel) entonces..."
    .List(8) = "        ZBuffer(pixel) = Z"
    .List(9) = "        Frame(pixel) = Color del pixel"
    .List(10) = "      Fin Si"
    .List(11) = "    Siguiente pixel"
    .List(12) = "  Siguiente cara"
    .List(13) = "Fin del ALGORITMO Z-BUFFER"

  End With

End Sub
```

```
Private Sub InicEcuPlanos()

' Inicialización del cuadro de las ecuaciones de los planos
'   visibles
' ----------------------------------------------------------------

  '--- rellena las cabeceras del cuadro de las ecuaciones de
  '      planos visibles

  With grdEcuPlanos

    .Rows = 1

    .ColWidth(KbNP) = GiTPC * 6: _
     .TextMatrix(0, KbNP) = "Plano"                        'columna Plano

    .ColWidth(KbA) = GiTPC * 13: _
     .TextMatrix(0, KbA) = "Coeficiente A"                'columna Coef.A

    .ColWidth(KbB) = .ColWidth(KbA): _
     .TextMatrix(0, KbB) = "Coeficiente B"                'columna Coef.B

    .ColWidth(KbC) = .ColWidth(KbA): _
     .TextMatrix(0, KbC) = "Coeficiente C"                'columna Coef.C

    .ColWidth(KbD) = GiTPC * 16: _
     .TextMatrix(0, KbD) = "Coeficiente D"                'columna Coef.D

    .ColWidth(KbCP) = .ColWidth(KbNP): _
     .TextMatrix(0, KbCP) = "Color"                        'columna Color

  End With

End Sub
```

```
Private Sub InicEjecAuto()

' Inicialización para la ejecución automática del algoritmo
'   Z-Buffer
' ----------------------------------------------------------------

  '--- si se pulsó el botón Automática o si se hizo click en el
  '     menú Ejecución/Automática...

  If GtBoton = "A" Or GtMenu = "A" Then
    GtEjec = "A"                    'activa la ejecución automática
    mnuEjecAuto.Checked = True          'marca la opción Automática
    mnuEjecMan.Checked = False           'desmarca la opción Manual

    '--- solamente si se pulsó el botón Automática...

    If GtBoton = "A" Then
      GbLA = 0: lstCodigo.Selected(GbLA) = True  'marca la primera
                                                 'línea del pseudocódigo
    End If

    tmrPaso.Enabled = True          'habilita el contador de tiempo
    Call EjecAuto                   'inicia la ejecución automática

  End If

End Sub
```

```
Private Sub InicEjecMan()

' Inicialización para la ejecución manual del algoritmo Z-Buffer
' -----------------------------------------------------------------

  '--- si se pulsó el botón Manual o si se hace click en el menú
  '      Ejecución/Manual...

  If GtBoton = "M" Or GtMenu = "M" Then
    GtEjec = "M"                        'activa la ejecución manual
    mnuEjecAuto.Checked = False      'desmarca la opción Automática
    mnuEjecMan.Checked = True               'marca la opción Manual

    '--- solamente si se pulsó el botón Manual...

    If GtBoton = "M" Then
      GbLA = 0: lstCodigo.Selected(GbLA) = True  'marca la primera
                                              'línea del pseudocódigo
    End If

    tmrPaso.Enabled = False     'deshabilita el contador de tiempo
    lstCodigo.SetFocus               'pasa el foco al pseudocódigo

  End If

End Sub
```

```
Private Sub InicMatFrame()

' Inicialización de la matriz Frame()
' ------------------------------------------------------------------

  Dim bI As Byte, bJ As Byte                 'contadores de bucles

  Screen.MousePointer = vbHourglass   'cambia el cursor a un reloj

  With grdBuffers

    '--- para cada zbpixel del cuadro, rellena el frame-buffer

    .Col = 4

    For bJ = 0 To 31
      For bI = 0 To 31
        .Row = (32 * bJ + bI) + 1
        GTbBuf.bFrame(bI, bJ) = 15
        .CellBackColor = vbWhite
      Next bI
    Next bJ

  End With

  Screen.MousePointer = vbDefault    'cambia el cursor al original

End Sub
```

```
Private Sub InicMatZBuf()

' Inicialización de la matriz ZBuffer()
' -----------------------------------------------------------------

  Dim bI As Byte, bJ As Byte                 'contadores de bucles

  Screen.MousePointer = vbHourglass   'cambia el cursor a un reloj

  With grdBuffers

    '--- para cada zbpixel del cuadro, rellena el Z-Buffer

    For bJ = 0 To 31
      For bI = 0 To 31
        .Row = (32 * bJ + bI) + 1
        GTbBuf.sZBuf(bI, bJ) = GsMaxZ
        .TextMatrix(.Row, KbZB) = "infinito"
      Next bI
    Next bJ

  End With

  Screen.MousePointer = vbDefault    'cambia el cursor al original

End Sub


Private Sub InicResult()

' Inicialización del cuadro de resultado del Z-Buffer
' -----------------------------------------------------------------

  Dim bI As Byte                                  'contador de bucle

  picResult.Cls                       'limpia el cuadro de resultado
  picResult.Scale (0, 0)-(32, 32)  'escalado entre (0,0) y (32,32)
  picResult.ForeColor = QBColor(7)        'color gris para la malla

  '--- traza la cuadrícula de 32x32 casillas

  For bI = 1 To 31
    picResult.Line (0, bI)-(32, bI)
    picResult.Line (bI, 0)-(bI, 32)
  Next bI

End Sub
```

```
Private Function iSigPixel(iPixel As Integer) As Integer

' Cálculo del siguiente pixel
' ------------------------------------------------------------------

  Dim bX As Byte, bY As Byte                  'contadores de bucles

  '--- si el pixel está entre el primero y el último de la cara...

  If GiPP <= iPixel And iPixel < GiPU Then

    Do
      iPixel = iPixel + 1
      bY = CByte(Int(iPixel / 32))            'calcula Y del pixel
      bX = CByte(iPixel - 32 * bY)            'calcula X del pixel
    Loop Until GTbBuf.oCara(bX, bY) Or iPixel > GiPU

  End If

  iSigPixel = iPixel                   'devuelve el siguiente pixel

End Function
```

```
Private Sub LeeFichero()

' Lectura del fichero del modelo
' -----------------------------------------------------------------

  Dim bI As Byte, bL As Byte, bX As Byte     'contadores de bucles
  Dim sFicVal As Single                'valor de una línea del fichero
  Dim tFicKey As String                'clave de una línea del fichero
  Dim tFicLin As String                'texto de una línea del fichero

  GtErrTit = "Error: Lectura del Fichero"          'título de error

  Screen.MousePointer = vbHourglass   'cambia el cursor a un reloj

  '--- comienza la lectura del fichero

  Open dlgFichero.filename For Input As #1        'abre el fichero
  Input #1, tFicLin: tFicLin = Trim(UCase(tFicLin)) 'lee una línea

  '--- si el fichero no es válido, error

  If tFicLin <> "AZB" Then
    GtErrText = "Fichero no válido"             'mensaje del error
    GoTo gFallo                   'realiza el tratamiento del error
  End If

  With GTmMod

    '--- mientras no se llegue al final del fichero...

    Do While Not EOF(1)

      Input #1, tFicLin

      '--- mientras haya una línea en blanco...

      Do While tFicLin = "": Input #1, tFicLin: Loop

      tFicKey = Left(Trim(UCase(tFicLin)), 1)  'construye la clave
      sFicVal = Val(Right(tFicLin, _
       Len(tFicLin) - 1))                     'construye los valores

      '--- si la clave es "V", se lee número total de vértices

      If tFicKey = "V" Then

        .bNTV = CByte(sFicVal)   'calcula número total de vértices
        ReDim .sV3D(0 To .bNTV - 1, KbX To KbZ): bX = 0
```

```
'--- si la clave es "X", se leen coordenadas 3D de vértices

ElseIf tFicKey = "X" Then

  If bX < .bNTV Then
    .sV3D(bX, KbX) = sFicVal
    For bI = KbY To KbZ
      Input #1, tFicLin: sFicVal = Val(tFicLin)
      .sV3D(bX, bI) = sFicVal     'calcula coordenadas 3D de
                                                  'vértices
    Next bI
    bX = bX + 1
  End If

'--- si la clave es "C", se lee número total de caras

ElseIf tFicKey = "C" Then

  If bX = .bNTV Then
    .bNTC = CByte(sFicVal)    'calcula número total de caras
    ReDim .bLVC(0 To .bNTC - 1, 0 To 3): bL = 0
    ReDim .bCol(0 To .bNTC - 1)
  Else                     'si faltan vértices por leer, error
    GtErrText = "Faltan vértices (X)"     'mensaje del error
    GoTo gFallo           'realiza el tratamiento del error
  End If

'--- si la clave es "L", se leen listas de vértices por cada
'     cara

ElseIf tFicKey = "L" Then

  If bL < .bNTC Then
    .bLVC(bL, 0) = CByte(sFicVal)

    If .bLVC(bL, 0) > UBound(.bLVC, 2) - 1 Then
      ReDim Preserve .bLVC(0 To .bNTC - 1, _
       0 To .bLVC(bL, 0) + 1)
    End If

    Input #1, tFicLin: sFicVal = Val(tFicLin)
    .bLVC(bL, 1) = CByte(sFicVal)

    For bI = 2 To .bLVC(bL, 0)
      Input #1, tFicLin: sFicVal = Val(tFicLin)
      .bLVC(bL, bI) = CByte(sFicVal)      'calcula listas de
                                 'vértices por cada cara
    Next bI

    Input #1, tFicLin: sFicVal = Val(tFicLin)
    .bCol(bL) = CByte(sFicVal)          'color de cada cara
    bL = bL + 1

  End If
```

```
    '--- si la clave es "O", se leen coordenadas del observador

    ElseIf tFicKey = "O" Then

      If bL = .bNTC Then
        GTcCam.sObs(KbX) = sFicVal

        For bI = KbY To KbZ
          Input #1, tFicLin: sFicVal = Val(tFicLin)
          GTcCam.sObs(bI) = sFicVal  'calcula coordenadas 3D del
                                                    'observador
        Next bI

      Else                        'si faltan caras por leer, error
        GtErrText = "Faltan caras (L)"        'mensaje del error
        GoTo gFallo             'realiza el tratamiento del error
      End If

    '--- si la clave es "M", se leen coordenadas 3D del punto de
    '     mira

    ElseIf tFicKey = "M" Then

      GTcCam.sMir(KbX) = sFicVal

      For bI = KbY To KbZ
        Input #1, tFicLin: sFicVal = Val(tFicLin)
        GTcCam.sMir(bI) = sFicVal    'calcula coordenadas 3D del
                                                  'punto de mira
      Next bI

    '--- si hay otra clave, error

    Else

      GtErrText = "Clave no válida"           'mensaje del error
      GoTo gFallo               'realiza el tratamiento del error

    End If
  Loop

End With

'--- finaliza la lectura del fichero

Close #1                                      'cierra el fichero

GtFichTit = dlgFichero.FileTitle       'título del fichero en la
                                              'ventana principal
GoModelo = True                        'hay un modelo en memoria

fraModelo.Caption = dlgFichero.FileTitle  'título del cuadro del
                                                        'modelo
```

```
  frmAZB.Caption = KtAZBTit & _
   " (" & fraModelo.Caption & ")"  'título de la ventana principal

  Screen.MousePointer = vbDefault    'cambia el cursor al original

  GtMenu = ""                              'menú Ejecución inicial
  Call DibModelo                                 'dibuja el modelo

  Exit Sub

  '--- realiza el tratamiento de errores

gFallo:

  If MsgBox(GtErrText, KiErrTipo, GtErrTit) = vbOK Then      'error
    Close #1                                    'cierra el fichero
    GtFichTit = ""                      'no hay fichero en memoria
    GoModelo = False                     'no hay modelo en memoria
    mnuEjec.Enabled = False          'menú Ejecución no disponible
    mnuCamb.Enabled = False            'menú Cambios no disponible
    Screen.MousePointer = vbDefault  'cambia el cursor al original
    Exit Sub
  End If

End Sub
```

```
Private Function oVisible(bCara As Byte) As Boolean

' Comprobación de visibilidad de una cara, según Newell
' -----------------------------------------------------------------

  Dim bNV As Byte                      'número de vértices de la cara
  Dim bV As Byte                                'contador de vértices
  Dim bVA As Byte, bVS As Byte  'vértices actual y siguiente de la
                                                               'cara
  Dim sC As Single   'coeficiente C de la ecuación del plano de la
                                                               'cara

  With GTmMod

    '--- para cada vértice de la cara, calcula el coeficiente C

    ReDim .sEcP(.bNTC, KbA To KbD)
    sC = 0: bNV = .bLVC(bCara, 0)

    For bV = 1 To bNV
      bVA = .bLVC(bCara, bV)
      bVS = IIf(bV <> bNV, .bLVC(bCara, bV + 1), .bLVC(bCara, 1))
      sC = sC + (.sV3P(bVA, KbX) - .sV3P(bVS, KbX)) * _
       (.sV3P(bVA, KbY) + .sV3P(bVS, KbY))              'calcula C
    Next bV

    .sEcP(bCara, KbC) = sC
    oVisible = IIf(.sEcP(bCara, KbC) >= 0, _
     True, False)             'devuelve la condición de visibilidad

  End With

End Function
```

```
Private Sub Pseudocodigo()

' Recorrido por el pseudocódigo del algoritmo Z-Buffer
' ---------------------------------------------------------------

  '--- empieza el recorrido por el pseudocódigo

  With lstCodigo

    '--- [00] Inicio del ALGORITMO Z-BUFFER

    If .ListIndex = 0 And GbLA = .ListIndex Then
      Call CalcDimResult  'calcula el "viewport" y las coordenadas
                                                    'del resultado
      .ListIndex = 1: GbLA = .ListIndex    'actualiza la línea del
                                                     'pseudocódigo

    '--- [01]   Inicializar matriz ZBuffer()

    ElseIf .ListIndex = 1 And GbLA = .ListIndex Then

      Call InicMatZBuf         'inicializa la matriz del Z-Buffer
      .ListIndex = 2: GbLA = .ListIndex    'actualiza la línea del
                                                     'pseudocódigo

    '--- [02]   Inicializar matriz Frame()

    ElseIf .ListIndex = 2 And GbLA = .ListIndex Then

      Call InicMatFrame     'inicializa la matriz del frame-buffer
      GbCA = 0              'inicializa el contador de cara actual
      .ListIndex = 3: GbLA = .ListIndex    'actualiza la línea del
                                                     'pseudocódigo

    '--- [03]   Para cada cara del modelo...

    ElseIf .ListIndex = 3 And GbLA = .ListIndex Then

      If GbCA < GTmMod.bNTC Then
        Call DibCara(GbCA, 12)        'marca la cara actual (rojo)
        .ListIndex = 4
      Else                      'si es la última cara del modelo...
        .ListIndex = 12
      End If

      GbLA = .ListIndex       'actualiza la línea del pseudocódigo
```

```
'--- [04]     Comprobar visibilidad de la cara

ElseIf .ListIndex = 4 And GbLA = .ListIndex Then

  If oVisible(GbCA) Then             'si la cara es visible...
    Call CalcEcuPlano(GbCA)  'calcula la ecuación del plano de
                                                    'la cara
    Call CalcPixels(GbCA)      'calcula los pixels de la cara
    GiPA = GiPP                  'inicializa el pixel actual
    GbPY = CByte(Int(GiPA / 32))         'calcula Y del pixel
    GbPX = CByte(GiPA - 32 * GbPY)       'calcula X del pixel
    .ListIndex = 5
  Else                           'si la cara no es visible...
    .ListIndex = 12
  End If

  GbLA = .ListIndex       'actualiza la línea del pseudocódigo

'--- [05]     Para cada pixel de la cara...

ElseIf .ListIndex = 5 And GbLA = .ListIndex Then

  If GiPA <= GiPU Then
    grdBuffers.TopRow = GiPA + 1    'coloca pixel actual en la
                                             'parte superior
    .ListIndex = 6
  Else                    'si es el último pixel de la cara...
    .ListIndex = 11
  End If

  GbLA = .ListIndex       'actualiza la línea del pseudocódigo

'--- [06]       Calcular coordenada Z del pixel

ElseIf .ListIndex = 6 And GbLA = .ListIndex Then

  GsZPA = sZPixel(GbCA, GbPX, GbPY)       'calcula Z del pixel
  grdBuffers.Row = GiPA + 1
  grdBuffers.TextMatrix(grdBuffers.Row, KbZ) = _
   Format(GsZPA, "#,##0.000")
  .ListIndex = 7: GbLA = .ListIndex    'actualiza la línea del
                                               'pseudocódigo

'--- [07]       Si Z < ZBuffer(pixel) entonces...

ElseIf .ListIndex = 7 And GbLA = .ListIndex Then

  If GsZPA < GTbBuf.sZBuf(GbPX, GbPY) Then
    .ListIndex = 8
  Else
    .ListIndex = 10
  End If

  GbLA = .ListIndex       'actualiza la línea del pseudocódigo
```

```
'--- [08]         ZBuffer(pixel) = Z

ElseIf .ListIndex = 8 And GbLA = .ListIndex Then

  GTbBuf.sZBuf(GbPX, GbPY) = GsZPA
  grdBuffers.TextMatrix(grdBuffers.Row, KbZB) = _
   Format(GsZPA, "#,##0.000")
  .ListIndex = 9: GbLA = .ListIndex    'actualiza la línea del
                                                  'pseudocódigo

'--- [09]         Frame(pixel) = Color del pixel

ElseIf .ListIndex = 9 And GbLA = .ListIndex Then

  GTbBuf.bFrame(GbPX, GbPY) = GTmMod.bCol(GbCA)
  grdBuffers.CellBackColor = QBColor(GTmMod.bCol(GbCA) Mod 15)
  picResult.Line (CInt(GbPX), CInt(GbPY))- _
   (CInt(GbPX + 1), CInt(GbPY + 1)), _
   QBColor(GTmMod.bCol(GbCA) Mod 15), BF
  .ListIndex = 10: GbLA = .ListIndex   'actualiza la línea del
                                                  'pseudocódigo

'--- [10]       Fin Si

ElseIf .ListIndex = 10 And GbLA = .ListIndex Then

  .ListIndex = 11: GbLA = .ListIndex   'actualiza la línea del
                                                  'pseudocódigo

'--- [11]     Siguiente pixel

ElseIf .ListIndex = 11 And GbLA = .ListIndex Then

  If GiPA < GiPU Then
    GiPA = iSigPixel(GiPA)             'pasa al siguiente pixel
    GbPY = CByte(Int(GiPA / 32))           'calcula Y del pixel
    GbPX = CByte(GiPA - 32 * GbPY)         'calcula X del pixel
    .ListIndex = 5
  Else
    .ListIndex = 12
  End If

  GbLA = .ListIndex       'actualiza la línea del pseudocódigo
```

```
    '--- [12]   Siguiente cara

    ElseIf .ListIndex = 12 And GbLA = .ListIndex Then

      If GbCA < GTmMod.bNTC Then
        Call DibCara(GbCA, 8)     'marca la cara comprobada (gris)
        GbCA = GbCA + 1                  'pasa a la siguiente cara
        .ListIndex = 3
      Else
        .ListIndex = 13
      End If

      GbLA = .ListIndex       'actualiza la línea del pseudocódigo

    '--- [13] Fin del ALGORITMO Z-BUFFER

    ElseIf .ListIndex = 13 And GbLA = .ListIndex Then

      .ListIndex = 13: GbLA = .ListIndex   'actualiza la línea del
                                                     'pseudocódigo

    '--- si el foco no coincide con la línea actual, se devuelve a
    '     la posición correcta

    Else
      .ListIndex = GbLA    'devuelve la posición correcta del foco
    End If

  End With

End Sub



Private Function sZPixel(bCara As Byte, bX As Byte, bY As Byte) _
  As Single

' Cálculo de la componente Z de un pixel
' ----------------------------------------------------------------

  With GTmMod

    sZPixel = (-.sEcP(bCara, KbA) * bX - _
     .sEcP(bCara, KbB) * bY - .sEcP(bCara, KbD)) / _
     .sEcP(bCara, KbC)                      'calcula Z de un pixel

  End With

End Function
```

```
Private Sub Form_Load()

' Carga del formulario base e inicialización de variables y
'   controles
' ----------------------------------------------------------------

  '--- inicializa las variables globales

  With Screen
    GbFont = CByte((.Width / .TwipsPerPixelX) / 80)    'calcula el
                                                   'tamaño de las letras
    GiTPC = .Width / 100  'calcula el 1% del tamaño de la pantalla
  End With

  frmAutor.Show vbModal        'muestra la ventana de presentación
  GtFichTit = ""                         'no hay fichero en memoria
  GoModelo = False                        'no hay modelo en memoria

  '--- prepara el formulario base de la ventana principal

  With frmAZB
    .Left = 0
    .Top = 0
    .Width = Screen.Width
    .Height = Screen.Height
    .Caption = KtAZBTit
  End With

  '--- prepara el marco del resultado del Z-Buffer

  With fraResult
    .Left = 0
    .Top = 0
    .Width = GiTPC * 40
    .Height = GiTPC * 2 + .Width
    .Font.Size = GbFont
    .Caption = "Resultado del algoritmo Z-Buffer"
  End With

  '--- prepara el cuadro del resultado del Z-Buffer

  With picResult
    .Left = GiTPC * 0.5
    .Top = GiTPC * 2.5
    .Width = fraResult.Width - GiTPC
    .Height = .Width
    .ScaleMode = 0
  End With

  Call InicResult    'inicializa cuadro del resultado del Z-Buffer
```

```
'--- prepara el marco del contenido de los buffers

  With fraBuffers
    .Left = fraResult.Width
    .Top = 0
    .Width = GiTPC * 31
    .Height = fraResult.Height
    .Font.Size = GbFont
    .Caption = "Contenido de los buffers"
  End With

  '--- prepara la cuadrícula del contenido de los buffers

  With grdBuffers
    .Left = fraBuffers.Left + GiTPC * 0.5
    .Top = GiTPC * 2.5
    .Width = fraBuffers.Width - GiTPC
    .Height = fraBuffers.Height - GiTPC * 3
    .Font.Size = fraBuffers.Font.Size
    .Cols = 5
    .Rows = (32 * 32) + 1
  End With

  Call InicBuffers  'inicializa cuadro de contenido de los buffers

  '--- prepara el marco del pseudocódigo del algoritmo Z-Buffer

  With fraCodigo
    .Left = fraResult.Width + fraBuffers.Width
    .Top = 0
    .Width = GiTPC * 29
    .Height = frmAZB.ScaleHeight - GiTPC * 31
    .Font.Size = GbFont
    .Caption = "Pseudocódigo Z-Buffer"
  End With

  '--- prepara el listado del pseudocódigo del algoritmo Z-Buffer

  With lstCodigo
    .Left = fraCodigo.Left + GiTPC * 0.5
    .Top = GiTPC * 2.5
    .Width = fraCodigo.Width - GiTPC
    .Height = fraCodigo.Height - GiTPC * 3
    .Font.Size = fraCodigo.Font.Size
  End With

  Call InicCodigo          'inicializa cuadro del pseudocódigo del
                                               'algoritmo Z-Buffer
```

```
  '--- prepara el marco de las ecuaciones de los planos visibles

  With fraEcuPlanos
    .Left = 0
    .Top = fraResult.Height
    .Width = fraResult.Width + fraBuffers.Width
    .Height = frmAZB.ScaleHeight - .Top
    .Font.Size = GbFont
    .Caption = "Ecuaciones de los planos visibles"
  End With

  '--- prepara la cuadrícula de las ecuaciones de los planos
  '     visibles

  With grdEcuPlanos
    .Left = GiTPC * 0.5
    .Top = fraEcuPlanos.Top + GiTPC * 2.5
    .Width = fraEcuPlanos.Width - GiTPC
    .Height = fraEcuPlanos.Height - GiTPC * 3
    .Font.Size = fraEcuPlanos.Font.Size
    .Cols = 6
    .Rows = 1
  End With

  Call InicEcuPlanos       'inicializa cuadro de las ecuaciones de
                                                'planos visibles

  '--- prepara el marco del modelo

  With fraModelo
    .Left = fraCodigo.Left
    .Top = fraCodigo.Height
    .Width = fraCodigo.Width
    .Height = frmAZB.ScaleHeight - .Top
    .Font.Size = GbFont
    .Caption = KtModTit
  End With

  '--- prepara el cuadro del modelo

  With picModelo
    .Left = fraModelo.Left + GiTPC * 0.5
    .Top = fraModelo.Top + GiTPC * 2.5
    .Width = fraModelo.Width - GiTPC
    .Height = .Width
    .ScaleMode = 0
  End With
  picModelo.Scale (0, 0)-(319, 319)

End Sub
```

```
Private Sub grdBuffers_GotFocus()

' Si la cuadrícula del contenido de los buffers tiene el foco...
' ----------------------------------------------------------------

  lstCodigo.SetFocus              'se pasa el foco al pseudocódigo

End Sub



Private Sub grdEcuPlanos_GotFocus()

' Si la cuadrícula de las ecuaciones de los planos visibles tiene
'   el foco...
' ----------------------------------------------------------------

  lstCodigo.SetFocus              'se pasa el foco al pseudocódigo

End Sub



Private Sub lstCodigo_KeyPress(KeyAscii As Integer)

' Si se pulsa una tecla en el pseudocódigo...
' ----------------------------------------------------------------

  '--- si hay un modelo en memoria...

  If GoModelo Then

    '--- si está activa la ejecución manual y la tecla es Intro...

    If GtEjec = "M" And KeyAscii = vbKeyReturn Then
      Call Pseudocodigo      'recorre el pseudocódigo del Z-Buffer
    End If

    '--- si está activa la ejecución automática y la tecla es
    '     Espacio...

    If GtEjec = "A" And KeyAscii = vbKeySpace Then
      tmrPaso.Enabled = Not tmrPaso.Enabled       'des/habilita el
                                                  'contador de tiempo
      Call Pseudocodigo      'recorre el pseudocódigo del Z-Buffer
    End If

  End If

End Sub
```

```
Private Sub mnuAyudAutor_Click()

' Si se hace click en el menú Ayuda/Autor [F8]...
' -----------------------------------------------------------------

  frmAutor.Show vbModal               'muestra la ventana de Autor

End Sub



Private Sub mnuAyudCont_Click()

' Si se hace click en el menú Ayuda/Contenido [F1]...
' -----------------------------------------------------------------

  frmAyuda.Show vbModeless            'muestra la ventana de Ayuda

End Sub



Private Sub mnuCambDir_Click()

' Si se hace click en el menú Cambios/Dirección de mira... [F7]...
' -----------------------------------------------------------------

  GtCamb = "D"
  frmCamb.Caption = "Cambios en Dirección de mira"
  frmCamb.Show vbModal              'muestra la ventana de Cambios

  If GoCambio Then _
   Call DibModelo         'si hay cambio de componentes, dibuja el
                                                     'modelo nuevo

End Sub
```

```
Private Sub mnuCambMir_Click()

' Si se hace click en el menú Cambios/Punto de mira... [F6]...
' -----------------------------------------------------------------

  GtCamb = "P"
  frmCamb.Caption = "Cambios en Punto de mira"
  frmCamb.Show vbModal              'muestra la ventana de Cambios

  If GoCambio Then _
   Call DibModelo                   'si hay cambio de coordenadas,
                                           'dibuja el modelo nuevo

End Sub



Private Sub mnuCambObs_Click()

' Si se hace click en el menú Cambios/Observador... [F5]...
' -----------------------------------------------------------------

  GtCamb = "O"
  frmCamb.Caption = "Cambios en Observador"
  frmCamb.Show vbModal              'muestra la ventana de Cambios

  If GoCambio Then _
   Call DibModelo                   'si hay cambio de coordenadas,
                                           'dibuja el modelo nuevo

End Sub



Private Sub mnuEjecAuto_Click()

' Si se hace click en el menú Ejecución/Automática [F2]...
' -----------------------------------------------------------------

  '--- si está activa la ejecución manual...

  If GtEjec = "M" Then
    GtMenu = "A"    'se hizo click en el menú Ejecución/Automática
    Call InicEjecAuto          'inicializa la ejecución automática
  End If

End Sub
```

```
Private Sub mnuEjecMan_Click()

' Si se hace click en el menú Ejecución/Manual [F3]...
' ---------------------------------------------------------------

  '--- si está activa la ejecución automática...

  If GtEjec = "A" Then
    GtMenu = "M"        'se hizo click en el menú Ejecución/Manual
    Call DibModelo                             'redibuja el modelo
  End If

End Sub



Private Sub mnuFichAbrir_Click()

' Si se hace click en el menú Fichero/Abrir... [F4]...
' ---------------------------------------------------------------

  dlgFichero.Flags = KlFlags
  dlgFichero.ShowOpen                 'elige el fichero del modelo

'--- si se elige un modelo nuevo, se lee el fichero del modelo

  If GtFichTit <> dlgFichero.FileTitle Then Call LeeFichero

End Sub



Private Sub mnuFichSalir_Click()

' Si se hace click en el menú Fichero/Salir [F9]...
' ---------------------------------------------------------------

  Close #1                                      'cierra el fichero
  End                                        'finaliza el programa

End Sub



Private Sub picResult_GotFocus()

' Si el cuadro del resultado del Z-buffer tiene el foco...
' ---------------------------------------------------------------

  lstCodigo.SetFocus              'se pasa el foco al pseudocódigo

End Sub
```

```
Private Sub picResult_MouseMove(Button As Integer, _
  Shift As Integer, X As Single, Y As Single)

' Si se mueve el ratón en el cuadro del resultado del Z-Buffer...
' ----------------------------------------------------------------

  picResult.ToolTipText = CStr(Int(X)) & "," _
   & CStr(Int(Y))                          'muestra las coordenadas

End Sub



Private Sub tmrPaso_Timer()

' Activa el contador de tiempo para la ejecución automática del
'   algoritmo Z-Buffer
' ----------------------------------------------------------------

  tmrPaso.Interval = 1
  Call EjecAuto                     'inicia la ejecución automática

End Sub
```

A.3. FICHERO AZBAUTOR.FRM (FORM FRMAUTOR)

```
' aZb  Algoritmo Z-Buffer
' -----------------------
' Formulario: frmAutor (Autor).
' Autor: Antonio José Torres Rodríguez de Almansa.
' ==================================================================

Option Explicit



Private Sub Form_Load()

' Carga del formulario base e inicialización de variables y
'   controles
' ------------------------------------------------------------------

  '--- formulario base de la ventana de Autor

  With frmAutor
    .Left = 0
    .Top = 0
    .Width = 520 * Screen.TwipsPerPixelX + GiTPC * 0.75
    .Height = 300 * Screen.TwipsPerPixelY + GiTPC * 3
    .Caption = KtAZBTit
  End With

  '--- imagen de la ventana de Autor

  With imgAutor
    .Left = 0
    .Top = 0
    .Width = frmAutor.Width
    .Height = frmAutor.Height - GiTPC * 3
    .ToolTipText = "Haz click para continuar"
  End With

End Sub



Private Sub imgAutor_Click()

' Si se hace click sobre la imagen...
' ------------------------------------------------------------------

  Unload Me                       'descarga de la ventana de Autor

End Sub
```

A.4. FICHERO AZBAYUDA.FRM (FORM FRMAYUDA)

```
' aZb  Algoritmo Z-Buffer
' -----------------------
' Formulario: frmAyuda (Ayuda).
' Autor: Antonio José Torres Rodríguez de Almansa.
' ===================================================================

Option Explicit



Private Sub Form_Load()

' Carga del formulario base e inicialización de variables y
'   controles
' -------------------------------------------------------------------

  '--- formulario base de la ventana de Ayuda

  With frmAyuda
    .Left = 0
    .Top = 0
    .Width = GiTPC * 70
    .Height = Screen.Height
    .Caption = "Ayuda"
  End With

  '--- cuadro tipo Web de la ventana de Ayuda

  With webAyuda
    .Left = 0
    .Top = 0
    .Width = Me.Width
    .Height = Me.ScaleHeight
    .Navigate ("file://" & App.Path & _
     "/aZb.htm")                      'fichero de la página de ayuda
  End With

End Sub
```

A.5. Fichero aZbCamb.frm (form frmCamb)

```
' aZb  Algoritmo Z-Buffer
' -----------------------
' Formulario: frmCamb (Cambios).
' Autor: Antonio José Torres Rodríguez de Almansa.
' ==================================================================

Option Explicit



Private Sub cmdNo_Click()

' Si se hace click en el botón No...
' ------------------------------------------------------------------

  Unload Me                         'descarga de la ventana de Cambios

End Sub
```

```
Private Sub cmdSi_Click()

' Si se hace click en el botón Sí...
' -----------------------------------------------------------------

  Dim bI As Byte                                     'contador de bucle

  For bI = KbX To KbZ

    With txtCoord(bI)
      Select Case GtCamb

      '--- en caso de las coordenadas del observador...

        Case "O"
          If CSng(.Text) <> _
           GTcCam.sObs(bI) Then   'si hay cambio en coordenadas...
            GTcCam.sObs(bI) = CSng(.Text)          'se actualizan las
                                                         'coordenadas
            GoCambio = True       'se activa el indicador de cambio
          End If

      '--- en caso de las coordenadas del punto de mira...

        Case "P"
          If CSng(.Text) <> _
           GTcCam.sMir(bI) Then   'si hay cambio en coordenadas...
            GTcCam.sMir(bI) = CSng(.Text)          'se actualizan las
                                                         'coordenadas
            GoCambio = True       'se activa el indicador de cambio
          End If

      '--- en caso de las componentes de la dirección de mira...

        Case "D"
          If CSng(.Text) <> _
           GTcCam.sDir(bI) Then   'si hay cambio en componentes...
            GTcCam.sDir(bI) = CSng(.Text)          'se actualizan las
                                                         'componentes
            GoCambio = True       'se activa el indicador de cambio
          End If

      End Select

    End With
  Next bI

  Call cmdNo_Click             'igual que si se pulsa el botón "No"

End Sub
```

```
Private Sub Form_Load()

' Carga del formulario base e inicialización de variables y
'   controles
' ---------------------------------------------------------------

  Dim bI As Byte                                 'contador de bucles
  Dim iDif As Integer                        'diferencia entre casillas

  GoCambio = False           'todavía no hay cambio de coordenadas

  '--- formulario base de la ventana de Cambios

  With frmCamb
    .Left = 0
    .Top = 0
    .Width = GiTPC * 39
    .Height = GiTPC * 16
  End With

  iDif = GiTPC * 12
  For bI = KbX To KbZ

    '--- texto de las coordenadas

    With lblCoord(bI)
      .Left = GiTPC * 2 + bI * iDif
      .Top = GiTPC * 2
      .Width = GiTPC * 2
      .Height = GiTPC * 3
      .Font.Size = GbFont
    End With

    '--- valor de las coordenadas

    With txtCoord(bI)
      .Left = GiTPC * 4 + bI * iDif
      .Top = lblCoord(bI).Top
      .Width = GiTPC * 7
      .Height = lblCoord(bI).Height
      .Font.Size = lblCoord(bI).Font.Size

      Select Case GtCamb
        Case "O"         'en caso de coordenadas del observador...
          .Text = GTcCam.sObs(bI)
        Case "P"      'en caso de coordenadas del punto de mira...
          .Text = GTcCam.sMir(bI)
        Case "D"        'en caso de componentes de la dirección de
                                                         'mira...
          .Text = GTcCam.sDir(bI)
      End Select

    End With
```

```
    '--- dial de las coordenadas

    With updCoord(bI)
      .Left = GiTPC * 11 + bI * iDif
      .Top = lblCoord(bI).Top
      .Width = lblCoord(bI).Width
      .Height = lblCoord(bI).Height
    End With

  Next bI

  '--- botón "Sí"

  With cmdSi
    .Left = txtCoord(KbY).Left
    .Top = GiTPC * 7.5
    .Width = txtCoord(KbY).Width + updCoord(KbY).Width
    .Height = GiTPC * 4
    .Caption = "&Sí"
    .Font.Size = GbFont
  End With

  '--- botón "No"

  With cmdNo
    .Left = txtCoord(KbZ).Left
    .Top = cmdSi.Top
    .Width = cmdSi.Width
    .Height = cmdSi.Height
    .Caption = "&No"
    .Font.Size = cmdSi.Font.Size
  End With

End Sub
```

A.6. FICHERO AZBEJEC.FRM (FORM FRMEJEC)

```
' aZb  Algoritmo Z-Buffer
' -----------------------
' Formulario: frmEjec (Ejecución).
' Autor: Antonio José Torres Rodríguez de Almansa.
' ==================================================================

Option Explicit



Private Sub cmdAuto_Click()

' Si se hace click en el botón Automática...
' ------------------------------------------------------------------

  GtBoton = "A"                        'se pulsó el botón Automática
  Unload Me

End Sub



Private Sub cmdMan_Click()

' Si se hace click en el botón Manual...
' ------------------------------------------------------------------

  GtBoton = "M"                            'se pulsó el botón Manual
  Unload Me

End Sub
```

```
Private Sub Form_Load()

' Carga del formulario base e inicialización de variables y
'   controles
' ----------------------------------------------------------------

  '--- formulario base de la ventana de Ejecución

  With frmEjec
    .Left = 0
    .Top = 0
    .Width = GiTPC * 39
    .Height = GiTPC * 18
    .Caption = "Ejecución"
  End With

  '--- texto de la ventana de Ejecución

  With lblEjec
    .Left = GiTPC * 2
    .Top = GiTPC * 2
    .Width = GiTPC * 35
    .Height = GiTPC * 6
    .Font.Size = GbFont
    .Caption = "¿Qué tipo de ejecución deseas," & _
     vbNewLine & "Automática o Manual?"
  End With

  '--- botón "Automática"

  With cmdAuto
    .Left = GiTPC * 13
    .Top = GiTPC * 9
    .Width = GiTPC * 12
    .Height = GiTPC * 4
    .Font.Size = GbFont
    .Caption = "&Automática"
    .Cancel = True                               'botón de cancelación
    .Default = False
  End With

  '--- botón "Manual"

  With cmdMan
    .Left = GiTPC * 28
    .Top = cmdAuto.Top
    .Width = GiTPC * 9
    .Height = cmdMan.Height
    .Font.Size = cmdAuto.Font.Size
    .Caption = "&Manual"
    .Cancel = False
    .Default = True                                 'botón por defecto
  End With

End Sub
```

